



## Red Hat Decision Manager 7.12

Red Hat Decision Manager で Red Hat ビルドの  
OptaPlanner を使用した Solver の開発





## 法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

本書では、Red Hat Decision Manager で Red Hat ビルドの OptaPlanner を使用して Solver を開発する方法を説明し、計画問題に最適解を見つけ出します。

## 目次

前書き .....	6
多様性を受け入れるオープンソースの強化 .....	7
パート I. RED HAT ビルドの OPTAPLANNER プロジェクトの OPTAPLANNER 8 へのアップグレード .....	8
第1章 OPTAPLANNER 7.X 以前との互換性がない変更 .....	9
Java 11 以降が必要である	9
SolverFactory および PlannerBenchmarkFactory では KIE コンテナがサポートされなくなった	9
OSGi メタデータが削除された	9
Java シリアライゼーションを使用しなくなった	9
SolverFactory.getScoreDirectorFactory() が ScoreManager の後継に	9
SolverFactory: getSolverConfig() の削除	10
SolverConfig: buildSolver() の削除	10
PlannerBenchmarkConfig: buildPlannerBenchmark() の削除	10
SolverFactory: cloneSolverFactory() の削除	11
SolverFactory: createEmpty() の削除	11
XML <solver/> ルート要素が <a href="http://www.optaplanner.org/xsd/solver">http://www.optaplanner.org/xsd/solver</a> namespace namespace に所属するように	12
移動セクターの subPillarEnabled プロパティの削除	12
Solver: getScoreDirectorFactory() の削除	13
Solver.explainBestScore() has been removed	13
Solver インターフェイスのメソッド getBestSolution()、getBestScore() および getTimeMillisSpent() の削除	14
アノテーションスキンの削除	14
@PlanningFactProperty および @PlanningFactCollectionProperty の新規パッケージ	15
単一の filterClass が filterClassList の後継に	15
AcceptorConfig の名前が LocalSearchAcceptorConfig になった	16
カスタムプロパティ XML 設定形式の変更	17
<variableNameInclude/> 要素が <variableNameIncludes/> 要素 でラップされるように	17
ソリューションインターフェイス の削除	18
BestSolutionChangedEvent: isNewBestSolutionInitialized() の削除	19
<valueSelector>: variableName が属性に	20
パーティション検索: threadFactoryClass の削除	20
SimpleDoubleScore および HardSoftDoubleScore の削除	21
Score.toInitializedScore() の削除	21
さまざまな正当化 Comparators の削除	22
FeasibilityScore が の削除	22
@PlanningEntity.movableEntitySelectionFilter の削除	22
@PlanningVariable.reinitializeVariableEntityFilter の削除	23
*scoreHolder クラスがインターフェイスに	23
ValueRangeFactory クラスが final に	23
ConstraintMatchTotal と Indictment がインターフェイスに	23
scoreManager: 汎用型 Score の追加	24
ConstraintMatchTotal, ConstraintMatch and Indictment: 汎用型 Score の追加	24
ConstraintMatchAwareIncrementalScoreCalculator: 汎用型 Score の追加	25
AbstractCustomPhaseCommand の削除	26
スコア計算がパブリック API に移動	26
PlannerBenchmarkFactory: createFromSolverFactory() の削除	28
PlannerBenchmarkFactory: getPlannerBenchmarkConfig() の削除	28
XML <plannerBenchmark/> root 要素が <a href="http://www.optaplanner.org/xsd/benchmark">http://www.optaplanner.org/xsd/benchmark</a> namespace に属するように	29
ProblemBenchmarksConfig: xStreamAnnotatedClass の削除	29

BenchmarkAggregatorFrame.createAndDisplay(PlannerBenchmarkFactory) の削除	30
設定での JavaScript 式がサポート対象外に	31
非推奨の変数リスナーの削除	31
<b>第2章 OPTAPLANNER 8.2.0 と OPTAPLANNER 8.3.0 との間の変更</b>	<b>34</b>
ConstraintMatch.compareTo() が equals() と一貫性を持つように	34
<b>パート II. RED HAT ビルドの OPTAPLANNER のスタートガイド</b>	<b>35</b>
<b>第3章 RED HAT ビルドの OPTAPLANNER の概要</b>	<b>36</b>
3.1. 計画問題	36
3.2. 計画問題での NP 完全	36
3.3. 計画問題に対する解	37
3.4. 計画問題に対する制約	37
3.5. RED HAT ビルドの OPTAPLANNER で提供される例	38
3.6. N クィーン	41
3.7. クラウドバランシング	45
3.8. 巡回セールスマン (TSP - 巡回セールスマン問題)	45
3.9. テニスクラブのスケジュール	46
3.10. 会議のスケジュール	47
3.11. コースの時間割 (ITC 2007 TRACK 3 - カリキュラムのスケジュール)	48
3.12. マシンの再割当て (GOOGLE ROADEF 2012)	50
3.13. 配送経路	53
3.14. プロジェクトジョブのスケジュール	64
3.15. タスクの割り当て	66
3.16. 試験の時間割 (ITC 2007 TRACK 1 - 試験)	68
3.17. 看護師の勤務表 (INRC 2010)	71
3.18. 巡回トーナメント問題 (TTP)	76
3.19. コストを抑えるスケジュール	78
3.20. 投資資産クラスの割り当て (ポートフォリオの最適化)	81
3.21. 会議スケジュール	81
3.22. ロックツアー	84
3.23. 航空機乗組員のスケジューリング	84
<b>第4章 RED HAT ビルドの OPTAPLANNER の例のダウンロード</b>	<b>86</b>
4.1. OPTAPLANNER サンプルの実行	86
4.2. IDE (INTELLIJ, ECLIPSE、または NETBEANS) での RED HAT ビルドの OPTAPLANNER サンプルの実行	87
<b>第5章 BUSINESS CENTRAL での OPTAPLANNER のスタートガイド: 従業員勤務表の例</b>	<b>88</b>
5.1. BUSINESS CENTRAL への従業員勤務表サンプルプロジェクトのデプロイメント	88
5.2. 従業員の勤務表サンプルプロジェクトの再作成	88
5.3. REST API を使用した SOLVER へのアクセス	102
<b>第6章 OPTAPLANNER および QUARKUS の使用ガイド</b>	<b>107</b>
6.1. APACHE MAVEN および RED HAT ビルドの QUARKUS	107
6.2. MAVEN プラグインを使用した OPTAPLANNER RED HAT ビルドの QUARKUS MAVEN プロジェクトの作成	110
6.3. CODE.QUARKUS.REDHAT.COM を使用した RED HAT ビルドの QUARKUS MAVEN プロジェクトの作成	112
6.4. QUARKUS CLI を使用した RED HAT ビルドの QUARKUS MAVEN プロジェクトの作成	114
<b>パート III. RED HAT ビルドの OPTAPLANNER のソルバー</b>	<b>118</b>
<b>第7章 OPTAPLANNER ソルバーの REDHAT ビルドの設定</b>	<b>119</b>
7.1. XML ファイルを使用した OPTAPLANNER のソルバーの設定	119

7.2. JAVA API を使用した OPTAPLANNER のソルバーの設定	120
7.3. OPTAPLANNER アノテーション	121
7.4. OPTAPLANNER ドメインアクセスの指定	121
7.5. カスタムプロパティーの設定	122
<b>第8章 OPTAPLANNER ソルバー</b>	<b>123</b>
8.1. 問題の解決	123
8.2. ソルバー環境モード	124
8.3. OPTAPLANNER ソルバーのログレベルの変更	125
8.4. LOGBACK を使用して OPTAPLANNER ソルバーアクティビティをログに記録する	127
8.5. LOG4J を使用して OPTAPLANNER ソルバーアクティビティをログに記録する	128
8.6. ソルバーの監視	129
8.7. 乱数ジェネレーターの設定	132
<b>第9章 OPTAPLANNER SOLVERMANAGER</b>	<b>134</b>
9.1. 問題のバッチ解決	135
9.2. 解決して進捗状況を確認する	135
<b>パート IV. RED HAT ビルドの OPTAPLANNER クイックスタートガイド</b>	<b>137</b>
<b>第10章 RED HAT ビルドの QUARKUS 上の RED HAT ビルドの OPTAPLANNER: 時間割のクイックスタートガイド</b>	<b>138</b>
10.1. MAVEN プラグインを使用した OPTAPLANNER RED HAT ビルドの QUARKUS MAVEN プロジェクトの作成	139
10.2. ドメインオブジェクトのモデル化	141
10.3. 制約の定義およびスコアの計算	145
10.4. プランニングソリューションでのドメインオブジェクトの収集	148
10.5. SOLVER サービスの作成	150
10.6. ソルバー終了時間の設定	151
10.7. 時間割アプリケーションの実行	151
10.8. アプリケーションのテスト	152
10.9. ロギング	156
10.10. データベースを QUARKUS OPTAPLANNER 学校の時間割アプリケーションと統合する	156
10.11. MICROMETER と PROMETHEUS を使用して学校の時間割を監視する OPTAPLANNER QUARKUS アプリケーション	159
<b>第11章 RED HAT ビルドの QUARKUS での RED HAT ビルドの OPTAPLANNER: ワクチン接種予約スケジューラーのクイックスタートガイド</b>	<b>161</b>
11.1. OPTAPLANNER ワクチン接種予約のスケジューラーの仕組み	161
11.2. OPTAPLANNER ワクチン接種予約スケジューラーのダウンロードおよび実行	165
11.3. OPTAPLANNER ワクチン接種予約スケジューラーのパッケージ化および実行	166
11.4. OPTAPLANNER ワクチン接種予約スケジューラーのネイティブ実行可能ファイルとしての実行	166
11.5. 関連情報	167
<b>第12章 SPRING BOOT 上の RED HAT ビルドの OPTAPLANNER: 時間割のクイックスタートガイド</b>	<b>168</b>
12.1. SPRING BOOT 時間割のクイックスタートのダウンロードおよびビルド	169
12.2. ドメインオブジェクトのモデル化	169
12.3. 制約の定義およびスコアの計算	174
12.4. プランニングソリューションでのドメインオブジェクトの収集	176
12.5. TIMETABLE サービスの作成	179
12.6. ソルバー終了時間の設定	180
12.7. アプリケーションを実行可能にする手順	180
12.8. データベースと UI 統合の追加	184
12.9. MICROMETER と PROMETHEUS を使用して学校の時間割を監視する OPTAPLANNER SPRING BOOT アプリケーション	187

<b>第13章 OPTAPLANNER と JAVA の RED HAT ビルド: 学校の時間割のクイックスタートガイド</b>	<b>188</b>
13.1. MAVEN または GRADLE ビルドファイルを作成し、依存関係を追加する	189
13.2. ドメインオブジェクトのモデル化	192
13.3. 制約の定義およびスコアの計算	197
13.4. プランニングソリューションでのドメインオブジェクトの収集	199
13.5. TIMETABLEAPP.JAVA クラス	202
13.6. 学校の時間割アプリケーションを作成して実行する	206
13.7. アプリケーションのテスト	209
13.8. ロギング	213
13.9. MICROMETER と PROMETHEUS を使用して学校の時間割を監視する OPTAPLANNER JAVA アプリケーション	214
<b>パート V. RED HAT ビルドの OPTAPLANNER のスターターアプリケーション</b>	<b>216</b>
<b>第14章 IDE での RED HAT ビルドの OPTAPLANNER の使用: 従業員の勤務表サンプル</b>	<b>217</b>
14.1. 従業員勤務表スターターアプリケーションの概要	217
14.2. 従業員勤務表スターターアプリケーションの構築と実行	217
14.3. 従業員勤務表スターターアプリケーションのソースコードに関する概要	221
14.4. 従業員勤務表スターターアプリケーションの変更	223
<b>第15章 RED HAT OPENSIFT CONTAINER PLATFORM での RED HAT ビルドの OPTAPLANNER のデプロイメントおよび使用: 従業員勤務表スターターアプリケーション</b>	<b>225</b>
15.1. 従業員勤務表スターターアプリケーションの概要	225
15.2. OPENSIFT での従業員勤務表スターターアプリケーションのインストールおよび起動	225
15.3. 従業員勤務表スターターアプリケーションの使用	227
<b>第16章 RED HAT ビルドの OPTAPLANNER の配送経路プランニングスターターアプリケーションのデプロイおよび使用</b>	<b>237</b>
16.1. OPTAWEB 配送経路	237
16.2. OPTAWEB 配送経路デプロイメントファイルのダウンロードおよびビルド	238
16.3. RUNLOCALLY.SH スクリプトを使用してローカルで OPTAWEB 配送経路を実行します。	238
16.4. OPTAWEB 配送経路の手動での設定および実行	242
16.5. RED HAT OPENSIFT CONTAINER PLATFORM での OPTAWEB 配送経路の実行	244
16.6. OPTAWEB 配送経路の使用	246
16.7. OPTAWEB 配送経路の開発ガイド	248
16.8. OPTAWEB 配送経路のバックエンドアーキテクチャー	252
16.9. OPTAWEB 配送経路のバックエンド設定プロパティー	254
<b>付録A バージョン情報</b>	<b>256</b>
<b>付録B お問い合わせ先</b>	<b>257</b>





## 前書き

ビジネス上の意思決定の開発者は、OptaPlanner の Red Hat ビルドを使用して、計画の問題に対する最適なソリューションを決定するソルバーを開発できます。OptaPlanner は、Red Hat Decision Manager の組み込みコンポーネントです。Red Hat Decision Manager のサービスの一部として Solver を使用し、個別の制約のある限られたリソースを最適化することができます。

## 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みにより、これらの変更は今後の複数のリリースに対して段階的に実施されます。詳細は、[弊社の CTO である Chris Wright のメッセージ](#)を参照してください。

# パート I. RED HAT ビルドの OPTAPLANNER プロジェクトの OPTAPLANNER 8 へのアップグレード

OptaPlanner 7 以前のパブリック API で作成した OptaPlanner プロジェクトがあり、プロジェクトのコードを OptaPlanner 8 にアップグレードする場合は、本ガイドの情報を確認してください。本ガイドでは、パブリック API 以外の実装クラスへの変更も含まれています。

OptaPlanner パブリック API は、Java コードを介して OptaPlanner と対話できるようにする OptaPlanner ソースコードのサブセットです。同じメジャーリリース内で OptaPlanner の上位バージョンにアップグレードできるように、OptaPlanner は [セマンティックバージョンニング](#) に従います。これは、OptaPlanner のパブリック API を使用するコードを破損することなく、OptaPlanner 7.44 から OptaPlanner 7.48 にアップグレードできます。OptaPlanner パブリック API クラスは、OptaPlanner リリースのメジャーバージョン内で互換性があります。ただし、Red Hat が新しいメジャーバージョンをリリースした場合には、問題のある変更がパブリック API に導入される可能性があります。

OptaPlanner 8 は新しいメジャーリリースで、パブリック API への変更の一部は、以前のバージョンの OptaPlanner と互換性がありません。OptaPlanner 8 は、次の数年間における 8.x シリーズの基盤となります。このプロジェクトで長期的に利益が得られるように、以前のバージョンと互換性はなく、今回のリリースで必要とされる変更がパブリック API に加えられました。

表1 Red Hat Decision Manager および Red Hat ビルドの OptaPlanner

Decision Manager	OptaPlanner
7.7	7.33
7.8	7.39
7.9	7.44
7.10	7.48
7.11	8.5

各アップグレードノートにはラベルがつけられており、今回の変更がコードに影響を与える可能性を示します。以下の表は、各ラベルについて説明しています。

表2 アップグレードの影響に関するラベル

ラベル	影響
メジャー	コードに影響する可能性があります。
マイナー	特にコードを広範囲にカスタマイズしない限り、コードに影響を与えることはほぼありません。

以前のバージョンの OptaPlanner と互換性がない変更は、**パブリック API** タグでアノテーションが付けられます。

## 第1章 OPTAPLANNER 7.X 以前との互換性がない変更

本項に記載する変更は、OptaPlanner 7.x 以前のバージョンの OptaPlanner と互換性がありません。

### Java 11 以降が必要である メジャー、パブリック API

JRE または JDK 8 を使用している場合は、JDK 11 以上にアップグレードしてください。

- Linux の場合は、Linux ソフトウェアリポジトリから OpenJDK を取得します。Fedora および Red Hat Enterprise Linux の場合は、以下のコマンドを入力します。

```
sudo dnf install java-11-openjdk-devel
```

- Windows および macOS の場合は、[AdoptOpenJDK](#) の Web サイトから OpenJDK をダウンロードします。

### SolverFactory および PlannerBenchmarkFactory では KIE コンテナがサポートされなくなった メジャー、パブリック API

OptaPlanner は Kogito に準拠するため、KIE コンテナの概念は適用されません。そのため、**SolverFactory** では、KIE コンテナから **ソルバー** インスタンスを作成できなくなりました。これは、**PlannerBenchmarkFactory** およびベンチマークにも適用されます。

### OSGi メタデータが削除された メジャー、パブリック API

OSGi の使用量が制限されており、メンテナンスの負荷が高いため、OptaPlanner 8.x シリーズの OptaPlanner JAR ファイルでは、**META-INF/MANIFEST.MF** ファイルに OSGi メタデータが含まれなくなりました。

### Java シリアライゼーションを使用しなくなった マイナー、パブリック API

OptaPlanner 8 では、**Serializable** マーカーインターフェイスのほとんどがパブリック API で使用されなくなりました。JSON または別の形式でシリアライズすることを検討してください。

### SolverFactory.getScoreDirectorFactory() が ScoreManager の後継に メジャー、パブリック API

OptaPlanner のバージョン 7 では、スコアの指定に **ScoreDirectorFactory** を使用する必要があります。OptaPlanner のバージョン 8 では、新機能が **ScoreManager** に追加され、**ScoreDirector** の新規インスタンスを作成する理由がなくなりました。

OptaPlanner 7 の \*.java ファイルからの例:

```
ScoreDirectorFactory<CloudBalance> scoreDirectorFactory =
solverFactory.getScoreDirectorFactory();
try (ScoreDirector<CloudBalance> scoreDirector = scoreDirectorFactory.buildScoreDirector()) {
    scoreDirector.setWorkingSolution(solution);
    Score score = scoreDirector.calculateScore();
}
```

OptaPlanner 8 の \*.java ファイルからの例:

```
ScoreManager<CloudBalance> scoreManager = ScoreManager.create(solverFactory);
Score score = scoreManager.updateScore(solution);
```

**ScoreDirector** や **ScoreDirectorFactory** のインスタンスを取得できるようにするメソッドは、後継を導入せず、パブリック API から削除されました。**Score Director** インターフェイスの縮小バージョンがパブリック API にプロモートし、パブリック API に **ProblemFactChange** インターフェイスをプロモートしました。

### SolverFactory: **getSolverConfig()** の削除 マイナー、パブリック API

**SolverFactory.getSolverConfig()** メソッドが非推奨となり、**SolverFactory.create (SolverConfig)** メソッドが後継となりました。自然な流れになるように、**SolverConfig** インスタンスは、**SolverFactory** インスタンスよりも前にインスタンス化されるようになりました。以前の順序が削除されました。

OptaPlanner 7 の \*.java ファイルからの例:

```
SolverFactory<MySolution> solverFactory =
    SolverFactory.createFromXmlResource("../mySolverConfig.xml");
SolverConfig solverConfig = solverFactory.getSolverConfig();
...
Solver<MySolution> solver = solverFactory.buildSolver();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource("../mySolverConfig.xml");
...
SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
Solver<MySolution> solver = solverFactory.buildSolver();
```

**ClassLoader** も指定した場合には、**SolverConfig.createFromXmlResource()** と **SolverFactory.create()** の両方に指定するようにします。

### SolverConfig: **buildSolver()** の削除 マイナー、パブリック API

**SolverConfig.buildSolver()** メソッドは、パブリック API に属さない内部メソッドです。代わりに **SolverFactory.buildSolver()** メソッドを使用してください。

OptaPlanner 7 の \*.java ファイルからの例:

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource("../mySolverConfig.xml");
...
Solver<MySolution> solver = solverConfig.buildSolver();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource("../mySolverConfig.xml");
...
SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
Solver<MySolution> solver = solverFactory.buildSolver();
```

### PlannerBenchmarkConfig: **buildPlannerBenchmark()** の削除 マイナー、パブリック API

**PlannerBenchmarkConfig.buildPlannerBenchmark()** メソッドは、パブリック API に属さない内部メソッドです。代わりに **PlannerBenchmarkFactory.buildPlannerBenchmark()** メソッドを使用してください。

OptaPlanner 7 の \*.java ファイルからの例:

```
PlannerBenchmarkConfig benchmarkConfig = PlannerBenchmarkConfig.createFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
...
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
PlannerBenchmarkConfig benchmarkConfig = PlannerBenchmarkConfig.createFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
...
PlannerBenchmarkFactory benchmarkFactory =
    PlannerBenchmarkFactory.create(benchmarkConfig);
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

### **SolverFactory: cloneSolverFactory()** の削除 マイナー、パブリック API

**SolverFactory.cloneSolverFactory()** メソッドは非推奨となり、新しい **SolverConfig(SolverConfig)** のコピーコンストラクターが後継となり、**SolverFactory.cloneSolverFactory()** メソッドが削除されました。

OptaPlanner 7 の \*.java ファイルからの例:

```
private SolverFactory<MySolution> base;

public void userRequest(..., long userInput) {
    SolverFactory<MySolution> solverFactory = base.cloneSolverFactory();
    solverFactory.getSolverConfig()
        .getTerminationConfig()
        .setMinutesSpentLimit(userInput);
    Solver<MySolution> solver = solverFactory.buildSolver();
    ...
}
```

OptaPlanner 8 の \*.java ファイルからの例:

```
private SolverConfig base;

public void userRequest(..., long userInput) {
    SolverConfig solverConfig = new SolverConfig(base); // Copy it
    solverConfig.getTerminationConfig()
        .setMinutesSpentLimit(userInput);
    SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
    Solver<MySolution> solver = solverFactory.buildSolver();
    ...
}
```

### **SolverFactory: createEmpty()** の削除

## マイナー、パブリック API

**SolverFactory.createEmpty()** メソッドは非推奨となり、新しい **SolverConfig()** メソッドが後継となりました。 **SolverFactory.createEmpty()** メソッドが削除されました。

OptaPlanner 7 の \*.java ファイルからの例:

```
SolverFactory<MySolution> solverFactory = SolverFactory.createEmpty();
SolverConfig solverConfig = solverFactory.getSolverConfig()
...
Solver<MySolution> solver = solverFactory.buildSolver();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
SolverConfig solverConfig = new SolverConfig();
...
SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
Solver<MySolution> solver = solverFactory.buildSolver();
```

XML **<solver/>** ルート要素が **http://www.optaplanner.org/xsd/solver namespace namespace** に所属するように  
メジャー、パブリック API

OptaPlanner はソルバー設定に XML スキーマ定義を提供するようになりました。OptaPlanner は、既存の XML 設定との互換性を維持していますが、今後サポート対象となるのが有効な設定 XML のみになる可能性があるため、XSD への移行を強く推奨します。

OptaPlanner 7 の \***SolverConfig.xml** ファイルからの例:

```
<?xml version="1.0" encoding="UTF-8"?>
<solver>
...
</solver>
```

OptaPlanner 8 の \***SolverConfig.xml** ファイルからの例:

```
<?xml version="1.0" encoding="UTF-8"?>
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
...
</solver>
```

XSD を使用すると、設定の XML 要素の一部を並べ替える必要がある場合があります。IDE のコード補完を使用して、有効な XML に移行します。

移動セクターの **subPillarEnabled** プロパティの削除  
マイナー、パブリック API

**PillarSwapMoveSelector** と **PillarChangeMoveSelector** にある **subPillarEnabled** プロパティが非推奨となり、新しいプロパティ **subPillarType** が後継となりました。 **subPillarEnabled** プロパティが削除されました。



OptaPlanner 7 の **\*SolverConfig.xml** および **\*BenchmarkConfig.xml** ファイルからの例:

```
<pillar...MoveSelector>
...
<pillarSelector>
  <subPillarEnabled>>false</subPillarEnabled>
  ...
</pillarSelector>
...
</pillar...MoveSelector>
```

OptaPlanner 8 の **\*SolverConfig.xml** および **\*BenchmarkConfig.xml** ファイルからの例:

```
<pillar...MoveSelector>
  <subPillarType>NONE</subPillarType>
  <pillarSelector>
    ...
  </pillarSelector>
  ...
</pillar...MoveSelector>
```

### Solver: **getScoreDirectorFactory()** の削除 メジャー、パブリック API

**getScoreDirectorFactory()** メソッドが非推奨となり、**Solver** と **SolverFactory** クラスの両方から削除されました。

UI でのスコアの計算または説明だけに **Solver** インスタンスを作成する必要がなくなりました。代わりに **ScoreManager** API を使用します。

OptaPlanner 7 の **\*.java** ファイルからの例:

```
SolverFactory<VehicleRoutingSolution> solverFactory = SolverFactory.createFromXmlResource(...);
Solver<VehicleRoutingSolution> solver = solverFactory.buildSolver();
uiScoreDirectorFactory = solver.getScoreDirectorFactory();
...
```

OptaPlanner 8 の **\*.java** ファイルからの例:

```
SolverFactory<VehicleRoutingSolution> solverFactory = SolverFactory.createFromXmlResource(...);
ScoreManager<VehicleRoutingSolution> scoreManager = ScoreManager.create(solverFactory);
...
```

**ScoreDirectorFactory** は、パブリック API の外部に常に配置され、全機能がそのパブリック API のさまざまな部分に公開されるので、使用しないようにしてください。

### Solver.**explainBestScore()** has been removed メジャー、パブリック API

**Solver** インターフェイスの **explainBestScore()** メソッドは 7.x で非推奨になり、削除されました。新しい **ScoreManager** API から同じ情報を取得できます。

Red Hat は、このメソッド呼び出しの結果をいかなる方法でも解析しないことを推奨します。

OptaPlanner 7 の \*.java ファイルからの例:

```
solver = ...;
scoreExplanation = solver.explainBestScore();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
MySolution solution = ...;
ScoreManager<MySolution> scoreManager = ...;
scoreExplanation = scoreManager.explainScore(solution);
```

## Solver インターフェイスのメソッド `getBestSolution()`、`getBestScore()` および `getTimeMillisSpent()` の削除 マイナー、パブリック API

**Solver** インターフェイスのメソッドが複数、7.x で非推奨になり、削除されました。**`Solver.addEventListener(...)`** で **`EventListener`** を登録すると、同じ情報を取得できます。

OptaPlanner 7 の \*.java ファイルからの例:

```
solver = ...;
solution = solver.getBestSolution();
score = solver.getBestScore();
timeMillisSpent = solver.getTimeMillisSpent();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
solver = ...;
solver.addEventListener(event -> {
    solution = event.getNewBestSolution();
    score = event.getNewBestScore();
    timeMillisSpent = event.getTimeMillisSpent();
});
```

## アノテーションスキンの削除 メジャー、パブリック API

ソルバー設定の `<scanAnnotatedClasses/>` ディレクティブが 7.x で非推奨となり、削除されました。

OptaPlanner 7 の \*.xml ファイルからの例:

```
<solver>
...
<scanAnnotatedClasses/>
...
</solver>
```

OptaPlanner 8 の \*.xml ファイルからの例:

```
<solver>
...
<solutionClass>...</solutionClass>
```

```
<entityClass>...</entityClass>
...
</solver>
```

## @PlanningFactProperty および @PlanningFactCollectionProperty の新規パッケージ メジャー、パブリック API

@PlanningFactProperty と @PlanningFactCollectionProperty のアノテーションは、@PlanningSolution など、他の同様のアノテーションと同じパッケージを共有するようになりました。古いアノテーションは 7.x で非推奨となり、削除されました。

OptaPlanner 7 の \*.java ファイルからの例:

```
import org.optaplanner.core.api.domain.solution.drools.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.solution.drools.ProblemFactProperty;
```

OptaPlanner 8 の \*.java ファイルからの例:

```
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.solution.ProblemFactProperty;
```

## 単一の filterClass が filterClassList の後継に マイナー、パブリック API

EntitySelector、ValueSelector および MoveSelector の設定は、設定 API とソルバー設定 XML の両方に単一のフィルタークラスが含まれるようになりました。

実際には、複数の選択フィルタークラスが必要ないため、全ロジックを実装する単一の選択フィルタークラスに置き換えることができます。1つの選択クラスを渡す場合に必要になるボイラープレートコードが少なくなります。

OptaPlanner 7 の \*.java ファイルからの例:

```
ValueSelectorConfig valueSelectorConfig = new ValueSelectorConfig();
valueSelectorConfig.setFilterClassList(Collections.singletonList(MySelectionFilterClass.class));
```

OptaPlanner 8 の \*.java ファイルからの例:

```
ValueSelectorConfig valueSelectorConfig = new ValueSelectorConfig();
valueSelectorConfig.setFilterClass(MySelectionFilterClass.class);
```

## 複数の選択フィルタークラスが1つの選択フィルタークラスに代わりました。

OptaPlanner 7 の \*.xml ファイルからの例:

```
<swapMoveSelector>
  <entitySelector>
    <filterClass>com.example.FilterA</filterClass>
    <filterClass>com.example.FilterB</filterClass>
  </entitySelector>
</swapMoveSelector>
```

OptaPlanner 7 の \*.java ファイルからの例:

```
package com.example;
...
public class FilterA implements SelectionFilter<MySolution, MyPlanningEntity> {

    @Override
    public boolean accept(ScoreDirector<MySolution> scoreDirector, MyPlanningEntity selection) {
        return selection.getValue() < 500;
    }
}
```

```
package com.example;
...
public class FilterB implements SelectionFilter<MySolution, MyPlanningEntity> {

    @Override
    public boolean accept(ScoreDirector<MySolution> scoreDirector, MyPlanningEntity selection) {
        return selection.getOrder() == Order.ASC;
    }
}
```

OptaPlanner 8 の \*.xml ファイルからの例:

```
<swapMoveSelector>
  <entitySelector>
    <filterClass>com.example.SingleEntityFilter</filterClass>
  </entitySelector>
</swapMoveSelector>
```

OptaPlanner 8 の \*.java ファイルからの例:

```
package com.example;
...
public class SingleEntityFilter implements SelectionFilter<MySolution, MyPlanningEntity> {

    @Override
    public boolean accept(ScoreDirector<MySolution> scoreDirector, MyPlanningEntity selection) {
        return selection.getValue() < 500 && selection.getOrder() == Order.ASC;
    }
}
```

## AcceptorConfig の名前が LocalSearchAcceptorConfig になった マイナー

これは設定 API にのみ影響します。Solver 設定 XML ファイルには影響はありません。

他の local-search 固有の設定クラスとの命名の一貫性が実装されました。

OptaPlanner 7 の \*.java ファイルからの例:

```
LocalSearchPhaseConfig localSearchPhaseConfig = new LocalSearchPhaseConfig()
    .withAcceptorConfig(new AcceptorConfig().withEntityTabuSize(5));
```

OptaPlanner 8 の \*.java ファイルからの例:

```
LocalSearchPhaseConfig localSearchPhaseConfig = new LocalSearchPhaseConfig()
    .withAcceptorConfig(new LocalSearchAcceptorConfig().withEntityTabuSize(5));
```

## カスタムプロパティ XML 設定形式の変更 マイナー、パブリック API

この問題は、特に

**<scoreDirectorFactory/>**、**<moveIteratorFactory/>**、**<moveListFactory/>**、**<partitionedSearch/>** および **<customPhase/>** XML などの ソルバー設定 XML にだけ影響を与えます。

この変更により、ビルド時に設定 XML の構造を有効にできるようになりました

OptaPlanner 7 の \*.xml ファイルからの例:

```
<partitionedSearch>
  <solutionPartitionerClass>com.example.MySolutionPartitioner</solutionPartitionerClass>
  <solutionPartitionerCustomProperties>
    <partCount>4</partCount> <!-- a custom property -->
    <minimumProcessListSize>300</minimumProcessListSize> <!-- a custom property -->
  </solutionPartitionerCustomProperties>
</partitionedSearch>
```

OptaPlanner 8 の \*.xml ファイルからの例:

```
<partitionedSearch>
  <solutionPartitionerClass>com.example.MySolutionPartitioner</solutionPartitionerClass>
  <solutionPartitionerCustomProperties>
    <property name="partCount" value="4"/> <!-- a custom property -->
    <property name="minimumProcessListSize" value="300"/> <!-- a custom property -->
  </solutionPartitionerCustomProperties>
</partitionedSearch>
```

## **<variableNameInclude/>** 要素が **<variableNameIncludes/>** 要素 でラップされるように マイナー、パブリック API

この更新は、特に **<swapMoveSelector/>** および **<pillarSwapMoveSelector/>** 要素のソルバー設定 XML にだけ影響があります。

この変更により、ビルド時に設定 XML の構造を有効にできるようになりました

OptaPlanner 7 の \*.xml ファイルからの例:

```
<swapMoveSelector>
  <variableNameInclude>variableA</variableNameInclude>
  <variableNameInclude>variableB</variableNameInclude>
</swapMoveSelector>
```

OptaPlanner 8 の \*.xml ファイルからの例:

```
<swapMoveSelector>
  <variableNameIncludes>
    <variableNameInclude>variableA</variableNameInclude>
  </variableNameIncludes>
```

```

<variableNameInclude>variableB</variableNameInclude>
</variableNameIncludes>
</swapMoveSelector>

```

## ソリューションインターフェイス の削除 マイナー、パブリック API

**Solution** インターフェイスは非推奨となり、削除されました。Business Central でだけ使用される **AbstractSolution** インターフェイスも削除されました。

**Solution** インターフェイスを削除し、**getScore()** メソッドに **@PlanningScore** のアノテーションを付け、**getProblemFacts()** メソッドは、問題ファクトゲッター (またはフィールド) に直接 **@ProblemFactCollectionProperty** アノテーションをつけるように置き換えます。

OptaPlanner 7 の \*.java ファイルからの例:

```

@PlanningSolution
public class CloudBalance implements Solution<HardSoftScore> {

    private List<CloudComputer> computerList;
    ...

    private HardSoftScore score;

    @ValueRangeProvider(id = "computerRange")
    public List<CloudComputer> getComputerList() {...}

    public HardSoftScore getScore() {...}
    public void setScore(HardSoftScore score) {...}

    public Collection<? extends Object> getProblemFacts() {
        List<Object> facts = new ArrayList<Object>();
        facts.addAll(computerList);
        ...
        return facts;
    }
}

```

OptaPlanner 8 の \*.java ファイルからの例:

```

@PlanningSolution
public class CloudBalance {

    private List<CloudComputer> computerList;
    ...

    private HardSoftScore score;

    @ValueRangeProvider(id = "computerRange")
    @ProblemFactCollectionProperty
    public List<CloudComputer> getComputerList() {...}

    @PlanningScore
    public HardSoftScore getScore() {...}
}

```

```

    public void setScore(HardSoftScore score) {...}

}

```

問題ファクト が1つで **コレクション** でラップされていない場合は、以下の例のように **@ProblemFactProperty** アノテーションを使用します。

OptaPlanner 7 の \*.java ファイルからの例:

```

@PlanningSolution
public class CloudBalance implements Solution<HardSoftScore> {

    private CloudParametrization parametrization;
    private List<CloudBuilding> buildingList;
    @ValueRangeProvider(id = "computerRange")
    private List<CloudComputer> computerList;
    ...

    public Collection<? extends Object> getProblemFacts() {
        List<Object> facts = new ArrayList<Object>();
        facts.add(parametrization); // not a Collection
        facts.addAll(buildingList);
        facts.addAll(computerList);
        ...
        return facts;
    }

}

```

OptaPlanner 8 の \*.java ファイルからの例:

```

@PlanningSolution
public class CloudBalance {

    @ProblemFactProperty
    private CloudParametrization parametrization;
    @ProblemFactCollectionProperty
    private List<CloudBuilding> buildingList;
    @ValueRangeProvider(id = "computerRange")
    @ProblemFactCollectionProperty
    private List<CloudComputer> computerList;
    ...

}

```

**@ProblemFactCollectionProperty** アノテーションがつけられたゲッター (またはフィールド) に **@PlanningEntityCollectionProperty** アノテーションを追加しないでください。

### BestSolutionChangedEvent: isNewBestSolutionInitialized() の削除

マイナー、パブリック API

**BestSolutionChangedEvent.isNewBestSolutionInitialized()** メソッドは非推奨となり、**BestSolutionChangedEvent.getNewBestSolution().getScore().isSolutionInitialized()** メソッドに置き換えられました。**BestSolutionChangedEvent.isNewBestSolutionInitialized()** メソッドが削除されました。

OptaPlanner 7 の \*.java ファイルからの例:

```
public void bestSolutionChanged(BestSolutionChangedEvent<CloudBalance> event) {
    if (event.isEveryProblemFactChangeProcessed()
        && event.isNewBestSolutionInitialized()) {
        ...
    }
}
```

OptaPlanner 8 の \*.java ファイルからの例:

```
public void bestSolutionChanged(BestSolutionChangedEvent<CloudBalance> event) {
    if (event.isEveryProblemFactChangeProcessed()
        && event.getNewBestSolution().getScore().isSolutionInitialized()) {
        ...
    }
}
```

**isFeasible()** をチェックする場合には、ソリューションが初期化されたかどうかを確認されます。

OptaPlanner 8 の \*.java ファイルからの例:

```
public void bestSolutionChanged(BestSolutionChangedEvent<CloudBalance> event) {
    if (event.isEveryProblemFactChangeProcessed()
        // isFeasible() checks isSolutionInitialized() too
        && event.getNewBestSolution().getScore().isFeasible()) {
        ...
    }
}
```

**<valueSelector>: variableName** が属性に  
マイナー、パブリック API

**<changeMoveSelector>** など、複数のプランニング変数と使用する場合の電源調整の移動セクター  
**<variableName>** XML 要素の後継として **variableName="..."** XML 属性が使用されるようになりました。この変更により、ソルバー設定の詳細レベルが軽減されました。7.x シリーズ全体で非推奨になり、以前の方法が削除されました。

OptaPlanner 7 の \***SolverConfig.xml** および \***BenchmarkConfig.xml** ファイルからの例:

```
<valueSelector>
  <variableName>room</variableName>
</valueSelector>
```

OptaPlanner 8 の \***SolverConfig.xml** および \***BenchmarkConfig.xml** ファイルからの例:

```
<valueSelector variableName="room"/>
```

パーティション検索: **threadFactoryClass** の削除  
マイナー、パブリック API

**<solver>** が **<threadFactoryClass>** 要素をサポートするようになってから時間が経っているので、**<partitionedSearch>** の **<threadFactoryClass>** 要素が削除されました。



OptaPlanner 7 の **\*SolverConfig.xml** および **\*BenchmarkConfig.xml** ファイルからの例:

```
<solver>
...
<partitionedSearch>
  <threadFactoryClass>...MyAppServerThreadFactory</threadFactoryClass>
...
</partitionedSearch>
</solver>
```

OptaPlanner 8 の **\*SolverConfig.xml** および **\*BenchmarkConfig.xml** ファイルからの例:

```
<solver>
  <threadFactoryClass>...MyAppServerThreadFactory</threadFactoryClass>
...
  <partitionedSearch>
    ...
  </partitionedSearch>
</solver>
```

### SimpleDoubleScore および HardSoftDoubleScore の削除 マイナー、パブリック API

スコアが破損する可能性があるため、二重ベースのスコアタイプを使用しないようにしてください。これらは削除されました。

OptaPlanner 7 の **\*.java** ファイルからの例:

```
@PlanningSolution
public class MyPlanningSolution {

    private SimpleDoubleScore score;

    ...

}
```

OptaPlanner 8 の **\*.java** ファイルからの例:

```
@PlanningSolution
public class MyPlanningSolution {

    private SimpleLongScore score;

    ...

}
```

### Score.toInitializedScore() の削除 マイナー、パブリック API

**Score.toInitializedScore()** メソッドは非推奨となり、7.x で **Score.withInitScore (int)** メソッドが後継となり、このメソッドは削除されました。

OptaPlanner 7 の \*.java ファイルからの例:

```
score = score.toInitializedScore();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
score = score.withInitScore(0);
```

## さまざまな正当化 **Comparators** の削除 マイナー、パブリック API

以下の **Comparator** 実装は 7.x で非推奨となり、削除されました。

- **org.optaplanner.core.api.score.comparator.NaturalScoreComparator**
- **org.optaplanner.core.api.score.constraint.ConstraintMatchScoreComparator**
- **org.optaplanner.core.api.score.constraint.ConstraintMatchTotalScoreComparator**
- **org.optaplanner.core.api.score.constraint.IndictmentScoreComparator**

OptaPlanner 7 の \*.java ファイルからの例:

```
NaturalScoreComparator comparator = new NaturalScoreComparator();
ConstraintMatchScoreComparator comparator2 = new ConstraintMatchScoreComparator();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
Comparator<Score> comparator = Comparable::compareTo;
Comparator<ConstraintMatch> comparator2 = Comparator.comparing(ConstraintMatch::getScore);
```

## **FeasibilityScore** が の削除 マイナー、パブリック API

**FeasibilityScore** インターフェイスは 7.x で非推奨となり、唯一の **isFeasible()** メソッドが **Score** supertype に移行されました。そのため、このインターフェイスが削除されました。

**Score** ではなく **HardSoftScore** などの、ultimate type で **Score** を参照するする必要があります。

## **@PlanningEntity.movableEntitySelectionFilter** の削除 マイナー、パブリック API

**@PlanningEntity** アノテーションの **movableEntitySelectionFilter** フィールドが 7.x で非推奨となり、新しいフィールド **pinningFilter** が導入されました。名前で、**@PlanningPin** アノテーションとの明確な関係が分かります。このフィルターは新しい **PinningFilter** インターフェイスを実装し、エンティティーが固定されている場合は true を、移動可能な場合は false を返します。そのため、この新しいフィルターのロジックは以前のフィルターと比較して反転されます。

ユーザーは **@PlanningEntity** アノテーションを更新し、以前のフィルターではなく新しいフィルターを指定する必要があります。以前のフィールドは削除されました。

OptaPlanner 7 の \*.java ファイルからの例:

```
@PlanningEntity(movableEntitySelectionFilter = MyMovableEntitySelectionFilter.class)
```

OptaPlanner 8 の \*.java ファイルからの例:

```
@PlanningEntity(pinningFilter = MyPinningFilter.class)
```

### @PlanningVariable.reinitializeVariableEntityFilter の削除 マイナー、パブリック API

@PlanningVariable アノテーションの **reinitializeVariableEntityFilter** フィールドは、7.x で非推奨になり、削除されました。

### \*scoreHolder クラスがインターフェイスに マイナー、パブリック API

OptaPlanner 7 では **ScoreHolder** クラスは、Drools スコア計算専用で使用されており、このクラスを使用する場合には、パブリックメソッドが複数公開されるので、ユーザーが意図的に破損させることも、スコアに悪影響を与えることも可能でした。

OptaPlanner 8 では、これらのメソッドが削除され、クラスがインターフェイスにバインドされています。多くのユーザーは、削除済みの手法や、危害を与える可能性のある手法を使用しません。

ただし、このような手法を使用していない場合には、スコアの説明と制約設定のエリアでパブリック API で代わりに適したものが検索できます。

### ValueRangeFactory クラスが final に マイナー

**ValueRangeFactory** クラスは、静的メソッドのみが含まれるファクトリークラスです。ユーザーがこのクラスを拡張する必要はないので、**final** となりました。

OptaPlanner 7 の \*.java ファイルからの例:

```
class MyValueRangeFactory extends ValueRangeFactory {
    ...
}
```

OptaPlanner 8 の \*.java ファイルからの例:

```
class MyValueRangeFactory {
    ...
}
```

### ConstraintMatchTotal と Indictment がインターフェイスに マイナー、パブリック API

**ConstraintMatchTotal** クラスと **Indictment** クラスがインターフェイスに変換されました。そのため、状態を変化できるようにするメソッドと合わせて、それらの実装がパブリック API から移行されました。このようなメソッドはパブリック API 用として作成されていないので、後継のメソッドはありません。

**ConstraintMatchAwareIncrementalScoreCalculator** を実装する場合は、インスタンス自体が必要になる場合があります。

```
ConstraintMatchTotal maximumCapacityMatchTotal = new ConstraintMatchTotal(...);
```

OptaPlanner 8 の \*.java ファイルからの例:

```
ConstraintMatchTotal maximumCapacityMatchTotal = new DefaultConstraintMatchTotal(...);
```

### scoreManager: 汎用型 Score の追加 メジャー、パブリック API

コード内でダウンキャストされないように、**ScoreManager** と **ScoreExplanation** API に汎用型 **Score** が含まれるようになりました (例: **Score** から **HardSoftScore**)。

OptaPlanner 7 の \*.java ファイルからの例:

```
@Inject // or @Autowired
ScoreManager<TimeTable> scoreManager;
```

OptaPlanner 8 の \*.java ファイルからの例:

```
@Inject // or @Autowired
ScoreManager<TimeTable, HardSoftScore> scoreManager;
```

OptaPlanner 7 の \*.java ファイルからの例:

```
ScoreExplanation<TimeTable> explanation = scoreManager.explainScore(timeTable);
HardSoftScore score = (HardSoftScore) explanation.getScore();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
ScoreExplanation<TimeTable, HardSoftScore> explanation =
scoreManager.explainScore(timeTable);
HardSoftScore score = explanation.getScore();
```

### ConstraintMatchTotal, ConstraintMatch and Indictment: 汎用型 Score の追加 メジャー

**ScoreManager** と **ScoreExplanation** と同様に、コード内でダウンキャストされないように、**ConstraintMatchTotal**、**ConstraintMatch** および **Indictment** API には汎用型 **Score** が含まれるようになりました (例: **Score** から **HardSoftScore**)。

OptaPlanner 7 の \*.java ファイルからの例:

```
ScoreExplanation<TimeTable> explanation = scoreManager.explainScore(timeTable);
Map<String, ConstraintMatchTotal> constraintMatchTotalMap =
scoreExplanation.getConstraintMatchTotalMap();
ConstraintMatchTotal constraintMatchTotal = constraintMatchTotalMap.get(constraintId);
HardSoftScore totalScore = (HardSoftScore) constraintMatchTotal.getScore();
```

OptaPlanner 8 の \*.java ファイルからの例:

```
ScoreExplanation<TimeTable, HardSoftScore> explanation =
scoreManager.explainScore(timeTable);
Map<String, ConstraintMatchTotal<HardSoftScore>> constraintMatchTotalMap =
scoreExplanation.getConstraintMatchTotalMap();
```

```

ConstraintMatchTotal<HardSoftScore> constraintMatchTotal =
constraintMatchTotalMap.get(constraintId);
HardSoftScore totalScore = constraintMatchTotal.getScore();

```

OptaPlanner 7 の \*.java ファイルからの例:

```

ScoreExplanation<TimeTable> explanation = scoreManager.explainScore(timeTable);
Map<Object, Indictment> indictamentMap = scoreExplanation.getIndictmentMap();
Indictment indictament = indictamentMap.get(lesson);
HardSoftScore totalScore = (HardSoftScore) indictament.getScore();

```

OptaPlanner 8 の \*.java ファイルからの例:

```

ScoreExplanation<TimeTable, HardSoftScore> explanation =
scoreManager.explainScore(timeTable);
Map<Object, Indictment<HardSoftScore>> indictamentMap = scoreExplanation.getIndictmentMap();
Indictment<HardSoftScore> indictament = indictamentMap.get(lesson);
HardSoftScore totalScore = indictament.getScore();

```

### ConstraintMatchAwareIncrementalScoreCalculator: 汎用型 Score の追加 マイナー

**ConstraintMatchTotal** and **Indictment** のロータイプを使用しないように、**ConstraintMatchAwareIncrementalScoreCalculator** インターフェイスに **Score** の汎用型パラメーターも含まれるようになりました。

OptaPlanner 7 の \*.java ファイルからの例:

```

public class MachineReassignmentIncrementalScoreCalculator
    implements ConstraintMatchAwareIncrementalScoreCalculator<MachineReassignment> {

    @Override
    public Collection<ConstraintMatchTotal> getConstraintMatchTotals() {
        ...
    }

    @Override
    public Map<Object, Indictment> getIndictmentMap() {
        ...
    }
}

```

OptaPlanner 8 の \*.java ファイルからの例:

```

public class MachineReassignmentIncrementalScoreCalculator
    implements ConstraintMatchAwareIncrementalScoreCalculator<MachineReassignment,
HardSoftLongScore> {

    @Override
    public Collection<ConstraintMatchTotal<HardSoftLongScore>> getConstraintMatchTotals() {
        ...
    }
}

```

```

@Override
public Map<Object, Indictment<HardSoftLongScore>> getIndictmentMap() {
    ...
}
}

```

## AbstractCustomPhaseCommand の削除 マイナー、パブリック API

抽象クラス **AbstractCustomPhaseCommand** が削除されました。拡張するクラスはいずれも **CustomPhaseCommand** インターフェイスを直接実装する必要があります。

OptaPlanner 7 の \*.java ファイルからの例:

```

public class DinnerPartySolutionInitializer extends AbstractCustomPhaseCommand<DinnerParty> {

    @Override
    public void changeWorkingSolution(ScoreDirector<DinnerParty> scoreDirector) {
        ...
    }
}

```

OptaPlanner 8 の \*.java ファイルからの例:

```

public class DinnerPartySolutionInitializer implements CustomPhaseCommand<DinnerParty> {

    @Override
    public void changeWorkingSolution(ScoreDirector<DinnerParty> scoreDirector) {
        ...
    }
}

```

## スコア計算がパブリック API に移動 メジャー

インターフェイス **EasyScoreCalculator**、**IncrementalScoreCalculator** および **ConstraintMatchAwareIncrementalScoreCalculator** がパブリック API の新規パッケージに移動されました。非推奨になったインターフェイスが削除されました。非推奨の **org.optaplanner.core.impl.score.director.incremental.AbstractIncrementalScoreCalculator** クラスも削除されました。パブリック API で、削除されたインターフェイスおよびクラスに対応するものに置き換えます。

OptaPlanner 7 の **EasyScoreCalculator.java** ファイルからの例:

```

...
import org.optaplanner.core.impl.score.director.easy.EasyScoreCalculator;
...

public class CloudBalancingEasyScoreCalculator implements EasyScoreCalculator<CloudBalance>

```

```
{
  ...
}
```

OptaPlanner 8 の **EasyScoreCalculator.java** ファイルからの例:

```
...
import org.optaplanner.core.api.score.calculator.EasyScoreCalculator;
...

public class CloudBalancingEasyScoreCalculator implements EasyScoreCalculator<CloudBalance,
HardSoftScore> {
  ...
}
```

OptaPlanner 7 の **IncrementalScoreCalculator.java** ファイルからの例:

```
...
import org.optaplanner.core.impl.score.director.incremental.AbstractIncrementalScoreCalculator;
...

public class CloudBalancingIncrementalScoreCalculator extends
AbstractIncrementalScoreCalculator<CloudBalance> {
  ...
}
```

OptaPlanner 8 の **IncrementalScoreCalculator.java** ファイルからの例:

```
...
import org.optaplanner.core.api.score.calculator.IncrementalScoreCalculator;
...

public class CloudBalancingIncrementalScoreCalculator implements
IncrementalScoreCalculator<CloudBalance, HardSoftScore> {
  ...
}
```

OptaPlanner 7 の **ConstraintMatchAwareIncrementalScoreCalculator.java** ファイルからの例:

```
...
import org.optaplanner.core.impl.score.director.incremental.AbstractIncrementalScoreCalculator;
import
org.optaplanner.core.impl.score.director.incremental.ConstraintMatchAwareIncrementalScoreCalculator
;
...

public class CheapTimeConstraintMatchAwareIncrementalScoreCalculator
  extends AbstractIncrementalScoreCalculator<CheapTimeSolution>
  implements ConstraintMatchAwareIncrementalScoreCalculator<CheapTimeSolution> {
  ...
}
```

OptaPlanner 8 の **ConstraintMatchAwareIncrementalScoreCalculator.java** ファイルからの例:

```

...
import org.optaplanner.core.api.score.calculator.ConstraintMatchAwareIncrementalScoreCalculator;
...

public class CheapTimeConstraintMatchAwareIncrementalScoreCalculator
    implements ConstraintMatchAwareIncrementalScoreCalculator<CheapTimeSolution,
HardMediumSoftLongScore> {
    ...
}

```

### PlannerBenchmarkFactory: createFromSolverFactory() の削除 メジャー、パブリック API

**PlannerBenchmarkFactory.createFromSolverFactory()** メソッドは非推奨となり、**PlannerBenchmarkFactory.createFromSolverConfigXmlResource (String)** メソッドが後継となりました。**PlannerBenchmarkFactory.createFromSolverFactory()** メソッドが削除されました。

OptaPlanner 7 の \*.java ファイルからの例:

```

SolverFactory<CloudBalance> solverFactory = SolverFactory.createFromXmlResource(
    ".../cloudBalancingSolverConfig.xml");
PlannerBenchmarkFactory benchmarkFactory =
    PlannerBenchmarkFactory.createFromSolverFactory(solverFactory);

```

OptaPlanner 8 の \*.java ファイルからの例:

```

PlannerBenchmarkFactory benchmarkFactory =
    PlannerBenchmarkFactory.createFromSolverConfigXmlResource(
        ".../cloudBalancingSolverConfig.xml");

```

ソルバー設定をプログラムで調節する場合に、**PlannerBenchmarkConfig.createFromSolverConfig (SolverConfig)** を使用してから代わりに **PlannerBenchmarkFactory.create (PlannerBenchmarkConfig)** を使用します。

### PlannerBenchmarkFactory: getPlannerBenchmarkConfig() の削除 マイナー、パブリック API

**PlannerBenchmarkFactory.getPlannerBenchmarkConfig()** メソッドは非推奨となり、**PlannerBenchmarkFactory.create (PlannerBenchmarkConfig)** メソッドが後継となりました。**PlannerBenchmarkConfig** インスタンスが **PlannerBenchmarkFactory** インスタンスよりも先にインスタンス化されるようになりました。この順序はより論理的です。**PlannerBenchmarkFactory.getPlannerBenchmarkConfig()** は削除されました。

OptaPlanner 7 の \*.java ファイルからの例:

```

PlannerBenchmarkFactory benchmarkFactory =
    PlannerBenchmarkFactory.createFromXmlResource(
        ".../cloudBalancingBenchmarkConfig.xml");
PlannerBenchmarkConfig benchmarkConfig = benchmarkFactory.getPlannerBenchmarkConfig();
...
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();

```

OptaPlanner 8 の \*.java ファイルからの例:



```

PlannerBenchmarkConfig benchmarkConfig = PlannerBenchmarkConfig.createFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
...
PlannerBenchmarkFactory benchmarkFactory =
    PlannerBenchmarkFactory.create(benchmarkConfig);
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();

```

### XML <plannerBenchmark/> root 要素が

<http://www.optaplanner.org/xsd/benchmark> namespace に属するように  
マイナー、パブリック API

OptaPlanner は、ベンチマーク設定の XML スキーマ定義 (XSD) を提供するようになりました。  
OptaPlanner は、既存の XML 設定との互換性を維持していますが、今後サポート対象となるのが有効な設定 XML のみになる可能性があるため、XSD への移行を強く推奨します。

OptaPlanner 7 の **\*BenchmarkConfig.xml** ファイルからの例:

```

<?xml version="1.0" encoding="UTF-8"?>
<plannerBenchmark>
...
</plannerBenchmark>

```

OptaPlanner 8 の **\*BenchmarkConfig.xml** ファイルからの例:

```

<?xml version="1.0" encoding="UTF-8"?>
<plannerBenchmark xmlns="https://www.optaplanner.org/xsd/benchmark"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/benchmark
        https://www.optaplanner.org/xsd/benchmark/benchmark.xsd">
...
</plannerBenchmark>

```

XSD を使用すると、設定の XML 要素の一部を並べ替える必要がある場合があります。IDE のコード補完を使用して、有効な XML に移行します。

### ProblemBenchmarksConfig: xStreamAnnotatedClass の削除

メジャー、パブリック API

<xStreamAnnotatedClass/> は、 **ProblemBenchmarksConfig** クラス内の、対応する **getXStreamAnnotatedClassList()** と **setXStreamAnnotatedClassList()** メソッドとともに、<problemBenchmarks/> 設定から削除されました。

OptaPlanner 7 の **\*.java** ファイルからの例:

```

ProblemBenchmarksConfig problemBenchmarksConfig = new ProblemBenchmarksConfig();
problemBenchmarksConfig.setXStreamAnnotatedClassList(MySolution.class);

```

OptaPlanner 8 の **\*.java** ファイルからの例:

```

package com.example;
...
public class MySolutionFileIO extends XStreamSolutionFileIO<MySolution> {
    public MySolutionFileIO() {
        super(MySolution.class);
    }
}

```

```

    }
}

...

ProblemBenchmarksConfig problemBenchmarksConfig = new ProblemBenchmarksConfig();
problemBenchmarksConfig.setSolutionFileIOClass(MySolutionFileIO.class);

```

OptaPlanner 7 の **\*BenchmarkConfig.xml** ファイルからの例:

```

<plannerBenchmark>
...
  <solverBenchmark>
    <problemBenchmarks>
      <xStreamAnnotatedClass>com.example.MySolution</xStreamAnnotatedClass>
      ...
    </problemBenchmarks>
    ...
  </solverBenchmark>
  ...
</plannerBenchmark>

```

OptaPlanner 8 の **\*BenchmarkConfig.xml** ファイルからの例:

```

<plannerBenchmark>
...
  <solverBenchmark>
    <problemBenchmarks>
      <!-- See the "After in *.java" section to create the MySolutionFileIO. -->
      <solutionFileIOClass>com.example.MySolutionFileIO</solutionFileIOClass>
      ...
    </problemBenchmarks>
    ...
  </solverBenchmark>
  ...
</plannerBenchmark>

```

## BenchmarkAggregatorFrame.createAndDisplay(PlannerBenchmarkFactory) の削除 マイナー

**BenchmarkAggregatorFrame.createAndDisplay (PlannerBenchmarkFactory)** メソッドは非推奨となり、**BenchmarkAggregatorFrame.createAndDisplayFromXmlResource (String)** メソッドが後継となりました。**BenchmarkAggregatorFrame.createAndDisplay (PlannerBenchmarkFactory)** メソッドが削除されました。

OptaPlanner 7 の **\*.java** ファイルからの例:

```

PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.createFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
BenchmarkAggregatorFrame.createAndDisplay(benchmarkFactory);

```

OptaPlanner 8 の **\*.java** ファイルからの例:

```
BenchmarkAggregatorFrame.createAndDisplayFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
```

ベンチマーク設定をプログラムで調整する場合は、代わりに **BenchmarkAggregatorFrame.createAndDisplay (PlannerBenchmarkConfig)** を使用できます。

設定での **JavaScript** 式がサポート対象外に  
マイナー

ソルバー設定とベンチマーク設定の両方に含まれるさまざまな要素では、ネストされた JavaScript 式をサポートしなくなりました。ユーザーは、自動設定または整数の定数のいずれかに置き換える必要があります。

OptaPlanner 7 の **solverConfig.xml** ファイルからの例:

```
<solver>
...
<moveThreadCount>availableProcessorCount - 1</moveThreadCount>
...
</solver>
```

OptaPlanner 8 の `solverConfig.xml` ファイルの例:

```
<solver>
...
<moveThreadCount>1</moveThreadCount> <!-- Alternatively, use "AUTO" or omit entirely. -->
...
</solver>
```

OptaPlanner 7 の **benchmarkConfig.xml** ファイルからの例:

```
<plannerBenchmark>
...
<parallelBenchmarkCount>availableProcessorCount - 1</parallelBenchmarkCount>
...
</plannerBenchmark>
```

OptaPlanner 8 の **benchmarkConfig.xml** ファイルからの例:

```
<plannerBenchmark>
...
<parallelBenchmarkCount>1</parallelBenchmarkCount> <!-- Alternatively, use "AUTO" or omit
entirely. -->
...
</plannerBenchmark>
```

非推奨の変数リスナーの削除  
メジャー、パブリック API

**org.optaplanner.core.impl.domain.variable.listener** パッケージから、非推奨の **StatefulVariableListener** インターフェイスと **VariableListenerAdapter** クラスと合わせて、の非推奨の **VariableListener** が削除されました。代わりに、**org.optaplanner.core.api.domain.variable** パッケージの **VariableListener** インターフェイスを使用してください。

OptaPlanner 7 の **VariableListener.java** ファイルからの例:

```
...
import org.optaplanner.core.impl.domain.variable.listener.VariableListenerAdapter;
...

public class MyVariableListener extends VariableListenerAdapter<Object> {

    ...

    @Override
    void afterEntityRemoved(ScoreDirector scoreDirector, Object entity);
    ...
}

...
}
```

OptaPlanner 8 の **VariableListener.java** ファイルからの例:

```
...
import org.optaplanner.core.api.domain.variable.VariableListener;
...

public class MyVariableListener extends VariableListener<MySolution, Object> {

    ...

    @Override
    void afterEntityRemoved(ScoreDirector<MySolution> scoreDirector, Object entity);
    ...
}

...
}
```

OptaPlanner 7 の **StatefulVariableListener.java** ファイルからの例:

```
...
import org.optaplanner.core.impl.domain.variable.listener.StatefulVariableListener;
...

public class MyStatefulVariableListener implements StatefulVariableListener<Object> {

    ...

    @Override
    public void clearWorkingSolution(ScoreDirector scoreDirector) {
        ...
    }

    ...
}
```

OptaPlanner 8 の **StatefulVariableListener.java** ファイルからの例:

```
...
import org.optaplanner.core.api.domain.variable.VariableListener;
...

public class MyStatefulVariableListener implements VariableListener<MySolution, Object> {

    ...

    @Override
    public void close() {
        ...
    }

    ...
}
```

## 第2章 OPTAPLANNER 8.2.0 と OPTAPLANNER 8.3.0 との間の変更

OptaPlanner 8.2.0 から OptaPlanner 8.3.0 の間で、本セクションに記載の変更が加えられました。

**ConstraintMatch.compareTo() が equals() と一貫性を持つように  
マイナー**

**ConstraintMatch** での **equals()** のオーバーライドが削除されました。そのため、2つの異なる **ConstraintMatch** インスタンスが同等とみなされなくなりました。これは、**compareTo()** メソッドとは対照的で、2つのインスタンスがすべて等しい場合に引き続き使用できます。

## パート II. RED HAT ビルドの OPTAPLANNER のスタートガイド

ビジネスルールの開発者は、Red Hat ビルドの OptaPlanner を使用して、限られたリソースや個別の制約の中で計画問題に対する最適解を見つけ出すことができます。

本書を使用して、Red Hat ビルドの OptaPlanner で Solver の開発を開始していきます。

## 第3章 RED HAT ビルドの OPTAPLANNER の概要

OptaPlanner は組み込み可能な軽量プランニングエンジンで、プランニングの問題を最適化します。最適化のためのヒューリスティック法およびメタヒューリスティック法を、非常に効率的なスコア計算と組み合わせ、一般的な Java プログラマーが計画問題を効率的に解決できるようにします。

たとえば、OptaPlanner は、さまざまなユースケースの解決に役立ちます。

- **従業員勤務表/患者一覧:** 看護師の勤務シフト作成を容易にし、病床管理を追跡します。
- **教育機関の時間割:** 授業、コース、試験、および会議の計画を容易にします。
- **工場の計画:** 自動車の組み立てライン、機械の操業計画、および作業員のタスク計画を追跡します。
- **在庫の削減:** 紙や金属などの資源の消費を減らし、無駄を最小限に抑えます。

どの組織も、制約のある限定されたリソース (従業員、資産、時間、および資金) を使用して製品やサービスを提供するといった計画問題に直面しています。

OptaPlanner は、Apache Software License 2.0 を使用するオープンソースソフトウェアです。100% Pure Java に認定されており、ほとんどの Java 仮想マシン (JVM) で稼働します。

### 3.1. 計画問題

**計画問題** では、限られたリソースや個別の制約の中で最適なゴールを見つけ出します。最適なゴールは、次のようなさまざまなものです。

- **最大の利益:** 最適なゴールにより、可能な限り高い利益が得られます。
- **経済活動の最小フットプリント:** 最適なゴールでは、環境負荷が最小となります。
- **スタッフ/顧客の最大満足:** 最適なゴールでは、スタッフ/顧客のニーズが優先されます。

これらのゴールに到達できるかどうかは、利用できるリソースの数に依存します。たとえば、以下のようリソースには制限があります。

- ユーザー数
- 時間
- 予算
- 装置、車両、コンピューター、施設などの物理資産

これらのリソースに関連する個別の制約についても考慮する必要があります。たとえば、要員が働くことのできる時間数、特定の装置を使用することのできる技能、または機器同士の互換性などです。

Red Hat ビルドの OptaPlanner は、Java プログラマーが制約の飽和性の問題を効率的に解決するのに役立ちます。最適化ヒューリスティックとメタヒューリスティックを効率的なスコア計算と組み合わせます。

### 3.2. 計画問題での NP 完全

例に挙げたユースケースは **通常 NP 完全または NP 困難** であり、以下のことが言えます。



- 問題に対する解を実用的な時間内に検証することが容易です。
- 問題に対する最適解を実用的な時間内に見つけ出す確実な方法がない。

この場合、一般的な 2 つの手法では不十分であるため、問題を解くのが予想より困難だと考えられます。

- 力まかせアルゴリズムでは (より高度な類似アルゴリズムであっても)、時間がかかり過ぎる。
- たとえば [ビンパッキング問題](#) のような迅速なアルゴリズムでは、**容量の大きい順でアイテムを入力すると、最適とはほど遠い解が返されます。**

高度な最適化アルゴリズムを用いる OptaPlanner であれば、このような計画問題に対する適切な解を、妥当な時間内に見つけ出すことができます。

### 3.3. 計画問題に対する解

計画問題には、多数の解が存在します。

以下に示すように、解は複数のカテゴリーに分類されます。

#### 可能解

可能解とは、制約に違反するかどうかは問わず、あらゆる解を指します。通常、計画問題には膨大な数の可能解が存在します。ただし、このような解の多くは、役に立ちません。

#### 実行可能解

実行可能解とは、いずれの (負の) ハード制約にも違反しない解を指します。実行可能解の数は、可能解の数に比例します。実行可能解が存在しないケースもあります。実行可能解は、可能解の部分集合です。

#### 最適解

最適解とは、最高スコアの解を指します。通常、計画問題には数個の最適解が存在します。実行可能解が存在せず、最適解が現実的ではない場合でも、計画問題には少なくとも 1 つの最適解が必ず存在します。

#### 見つけた最善解

最善解とは、指定された時間内に実施した検索で見つけた最高スコアの解を指します。通常、見つけた最善解は現実的で、十分な時間があれば最適解を見つけることができます。

直観には反していますが、小規模なデータセットの場合であっても、(正しく計算された場合は) 膨大な数の可能解が存在します。

**planner-engine** ディストリビューションディレクトリーのサンプルでも、ほとんどのインスタンスには多数の可能解が存在します。最適解を確実に見つけることができる方法は存在しないため、いかなる実行方法も、これらすべての可能解の部分集合を評価することしかできません。

膨大な数の可能解全体を効率的に網羅するために、OptaPlanner はさまざまな最適化アルゴリズムをサポートしています。

ユースケースによっては、ある最適化アルゴリズムが他のアルゴリズムより勝ることがありますが、それを事前に予測することは不可能です。OptaPlanner では、XML またはコード中の Solver 設定を数行変更するだけで、最適化アルゴリズムを切り替えることができます。

### 3.4. 計画問題に対する制約

通常、プランニングの問題には、少なくとも 2 つの制約レベルがあります。

- **(負の)ハード制約** は、絶対に違反してはならない。  
例: 1 人の教師は同時に 2 つの講義を受け持つことはできない。
- **(負の)ソフト制約** は、避けることが可能であれば違反してはならない。  
例: 教師 A は金曜日の午後に講義を受け持ちたくない。

正の制約を持つ問題もあります。

- **正のソフト制約 (ボーナス)** は、可能であれば満たす必要がある。  
例: 教師 B は月曜日の午前中に講義を受け持つことを希望している。

一部の基本的な問題にはハード制約のみがあります。問題によっては、3 つ以上の制約があります (例: ハード制約、中程度の制約、ソフト制約)。

これらの制約により、計画問題における **スコア計算方法** (または **適合度関数**) が定義されます。プランニングの問題の解は、それぞれスコアで等級付けすることができます。OptaPlanner のスコア制約は、Java などのオブジェクト指向言語または Drools ルールで記述されます。

このタイプのコードは柔軟で、スケーラビリティに優れます。

### 3.5. RED HAT ビルドの OPTAPLANNER で提供される例

Red Hat Decision Manager には、Red Hat ビルドの OptaPlanner のサンプルが複数同梱されています。たとえばコードなどを確認して、ニーズに合ったものに変更できます。



#### 注記

Red Hat は、Red Hat Decision Manager ディストリビューションに含まれるコードサンプルのサポートはしていません。

OptaPlanner サンプルには、教育関連のコンテストで出題された問題を解決するものもあります。以下の表の **Contest** 列には、このようなコンテストが掲載されています。また、コンテストの目的として、**現実的** か、**非現実的** かの識別をしています。**現実的なコンテスト** とは、以下の基準を満たす、独立した公式コンテストを指します。

- 明確に定義された実際のユースケースであること
- 実際に制約があること
- 実際のデータセットが複数あること
- 特定のハードウェアで特定の時間内に結果を再現できること
- 教育機関および/または企業の運用研究コミュニティが真剣に参加していること

現実的なコンテストでは、競合のソフトウェアや教育研究と OptaPlanner を客観的に比較できます。

表3.1 サンプルの概要

例	ドメイン	サイズ	コンテスト	ディレクトリー名
---	------	-----	-------	----------

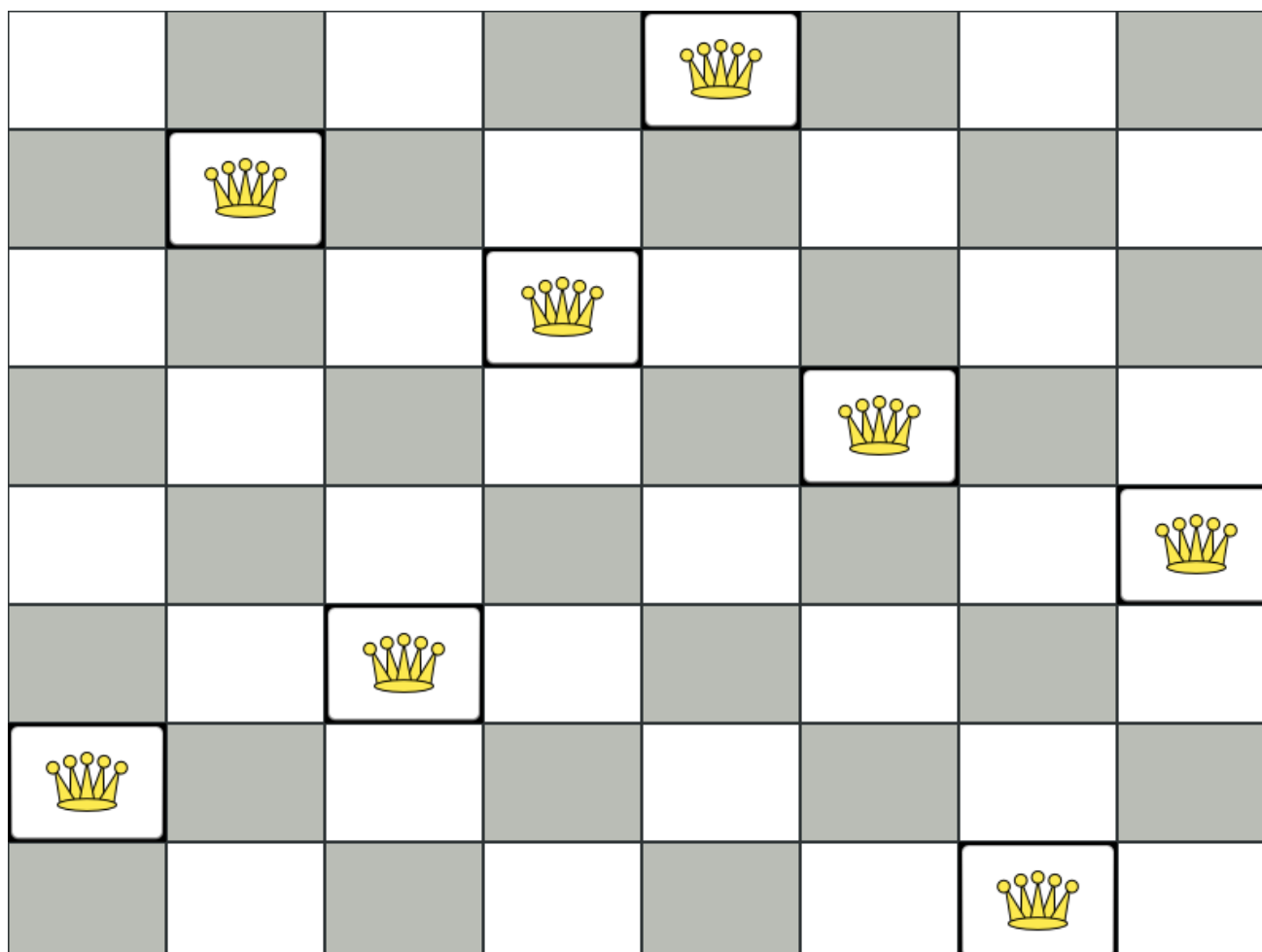
例	ドメイン	サイズ	コンテ スト	ディレクトリー名
N クイーン	エンティティークラス 1 つ (変数 1 つ)	エンティティークラス $\Leftarrow$ <b>256</b> 値 $\Leftarrow$ <b>256</b> 探索空間 $\Leftarrow$ <b><math>10^{616}</math></b>	無意味 (不正が可能)	<b>nqueens</b>
クラウド バラン シング	エンティティークラス 1 つ (変数 1 つ)	エンティティークラス $\Leftarrow$ <b>2400</b> 値 $\Leftarrow$ <b>800</b> 探索空間 $\Leftarrow$ <b><math>10^{6967}</math></b>	いいえ (弊社が定義)	<b>cloudbalancing</b>
巡回セ ールスマ ン	エンティティークラス 1 つ (連鎖変数 1 つ)	エンティティークラス $\Leftarrow$ <b>980</b> 値 $\Leftarrow$ <b>980</b> 探索空間 $\Leftarrow$ <b><math>10^{2504}</math></b>	現実的でない <a href="#">TSP Web</a>	<b>tsp</b>
テニス クラブの スケジ ュール	エンティティークラス 1 つ (変数 1 つ)	エンティティークラス $\Leftarrow$ <b>72</b> 値 $\Leftarrow$ <b>7</b> 探索空間 $\Leftarrow$ <b><math>10^{60}</math></b>	いいえ (弊社が定義)	<b>tennis</b>
会議の スケジ ュール	エンティティークラス 1 つ (変数 2 つ)	エンティティークラス $\Leftarrow$ <b>10</b> 値 $\Leftarrow$ <b>320</b> および $\Leftarrow$ <b>5</b> 探索空間 $\Leftarrow$ <b><math>10^{320}</math></b>	いいえ (弊社が定義)	<b>meetingscheduling</b>
コースの 時間割	エンティティークラス 1 つ (変数 2 つ)	エンティティークラス $\Leftarrow$ <b>434</b> 値 $\Leftarrow$ <b>25</b> および $\Leftarrow$ <b>20</b> 探索空間 $\Leftarrow$ <b><math>10^{1171}</math></b>	現実的 <a href="#">ITC 2007 track 3</a>	<b>curriculumCourse</b>
マシンの 再割当て	エンティティークラス 1 つ (変数 1 つ)	エンティティークラス $\Leftarrow$ <b>50000</b> 値 $\Leftarrow$ <b>5000</b> 探索空間 $\Leftarrow$ <b><math>10^{184948}</math></b>	ほぼ現実的 <a href="#">ROADEF 2012</a>	<b>machineReassignment</b>

例	ドメイン	サイズ	コンテ スト	ディレクトリー名
配送経路	エンティティークラス 1 つ (連鎖変数 1 つ)  シャドウエンティ ティークラス 1 つ  (自動シャドウ変数 1 つ)	エンティティークラス 1 つ 値 $\Leftarrow$ <b>2740</b> 値 $\Leftarrow$ <b>2795</b>  探索空間 $\Leftarrow$ <b><math>10^{8380}</math></b>	現実的で ない VRP Web	<b>vehiclerouting</b>
時間枠が ある中で の 配送経 路	配送経路すべて (シャドウ変数 1 つ)	エンティティークラス 1 つ 値 $\Leftarrow$ <b>2740</b> 値 $\Leftarrow$ <b>2795</b>  探索空間 $\Leftarrow$ <b><math>10^{8380}</math></b>	現実的で ない VRP Web	<b>vehiclerouting</b>
プロジェ クトジョ ブのスケ ジュール	エンティティークラス 1 つ (変数 2 つ)  (シャドウ変数 1 つ)	エンティティークラス 1 つ 値 $\Leftarrow$ <b>640</b> 値 $\Leftarrow$ ? および $\Leftarrow$ ?  探索空間 $\Leftarrow$ ?	ほぼ現実 的 <a href="#">MISTA 2013</a>	<b>projectjobschedulin g</b>
タスクの 割り当て	エンティティークラス 1 つ (連鎖変数 1 つ)  (シャドウ変数 1 つ)  シャドウエンティ ティークラス 1 つ  (自動シャドウ変数 1 つ)	エンティティークラス 1 つ 値 $\Leftarrow$ <b>500</b> 値 $\Leftarrow$ <b>520</b>  探索空間 $\Leftarrow$ <b><math>10^{1168}</math></b>	いいえ (弊 社が定義)	<b>taskassigning</b>
試験の時 間割	エンティティークラス 2 つ (同じ階層) (変数 2 つ)	エンティティークラス 2 つ 値 $\Leftarrow$ <b>1096</b> 値 $\Leftarrow$ <b>80</b> および $\Leftarrow$ <b>49</b>  探索空間 $\Leftarrow$ <b><math>10^{3374}</math></b>	現実的 <a href="#">ITC 2007 track 1</a>	<b>examination</b>
看護師の 勤務表	エンティティークラス 1 つ (変数 1 つ)	エンティティークラス 1 つ 値 $\Leftarrow$ <b>752</b> 値 $\Leftarrow$ <b>50</b>  探索空間 $\Leftarrow$ <b><math>10^{1277}</math></b>	現実的 <a href="#">INRC 2010</a>	<b>nurserostering</b>

例	ドメイン	サイズ	コンテス ト	ディレクトリー名
巡回トー ナメント	エンティティークラス1 つ  (変数1つ)	エンティティークラス $\Leftarrow$ <b>1560</b>  値 $\Leftarrow$ <b>78</b>  探索空間 $\Leftarrow$ <b><math>10^{2301}</math></b>	現実的で ない <a href="#">TTP</a>	<b>travelingtournament</b>
コストを 抑えるス ケジュー ル	エンティティークラス1 つ  (変数2つ)	エンティティークラス $\Leftarrow$ <b>500</b>  値 $\Leftarrow$ <b>100</b> および $\Leftarrow$ <b>288</b>  探索空間 $\Leftarrow$ <b><math>10^{20078}</math></b>	ほぼ現実 的 <a href="#">ICON</a> <a href="#">Energy</a>	<b>cheaptimeschedulin g</b>
投資	エンティティークラス1 つ  (変数1つ)	エンティティークラス $\Leftarrow$ <b>11</b>  値 = <b>1000</b>  探索空間 $\Leftarrow$ <b><math>10^4</math></b>	いいえ (弊 社が定義)	<b>investment</b>
会議スケ ジュール	エンティティークラス1 つ  (変数2つ)	エンティティークラス $\Leftarrow$ <b>216</b>  値 $\Leftarrow$ <b>18</b> および $\Leftarrow$ <b>20</b>  探索空間 $\Leftarrow$ <b><math>10^{552}</math></b>	いいえ (弊 社が定義)	<b>conferencescheduli ng</b>
ロックツ アー	エンティティークラス1 つ  (連鎖変数1つ)  (シャドウ変数4つ)  シャドウエンティ ティークラス1つ  (自動シャドウ変数1つ)	エンティティークラス $\Leftarrow$ <b>47</b>  値 $\Leftarrow$ <b>48</b>  探索空間 $\Leftarrow$ <b><math>10^{59}</math></b>	いいえ (弊 社が定義)	<b>rocktour</b>
航空機乗 組員のス ケジュー リング	エンティティークラス1 つ  (変数1つ)  シャドウエンティ ティークラス1つ  (自動シャドウ変数1つ)	エンティティークラス $\Leftarrow$ <b>4375</b>  値 $\Leftarrow$ <b>750</b>  探索空間 $\Leftarrow$ <b><math>10^{12578}</math></b>	いいえ (弊 社が定義)	<b>flightcrewschedulin g</b>

### 3.6. N キーン

$n$  サイズのチェスボードに、他のキーンに取られないキーンに  $n$  個のキーンを置きます。最も一般的な  $n$  キーンパズルは、 $n = 8$  の 8 個のキーンパズルです。



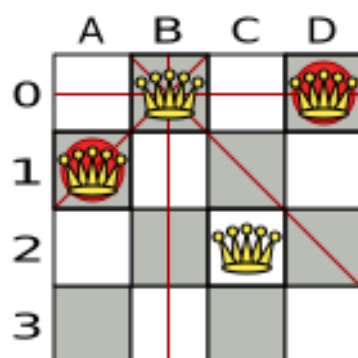
制約:

- $n$  列および  $n$  行のチェスボードを使用します。
- チェスボードに  $n$  個のクイーンを置きます。
- クイーンが他のクイーンに取られないように配置します。クイーンは、同じ水平線上、垂直線上、対角線上にある他のクイーンを取ることができます。

本書では、4 つのクイーンパズルを主な例として多用しています。

以下が提案された解です。

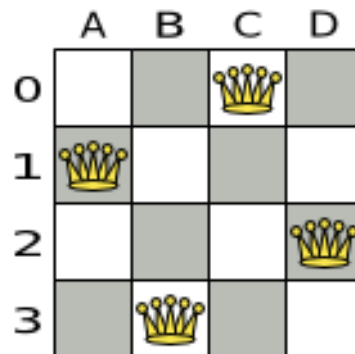
図3.14 個のクイーンパズルの誤った解



上記の解は、**A1** と **B0** (および **B0** と **D0**) のクイーンがお互いに駒を取れるので間違っています。**B0** のクイーンをどこに置けば他のクイーンに取られないようにするという制約は順守できますが、**n** 個のクイーンを置くという制約に違反します。

以下は正しい解です。

図3.2 クイーン 4 個のパズルの正しい解



すべての制約が満たされているので、これが正解です。

**n** クイーンパズルでは、正解が複数存在する場合があります。特定の **n** に対して考えられる解を見つけるのではなく、特定の **n** に対する正しい解を1つ導き出すことにフォーカスします。

## 問題の規模

```
4queens has 4 queens with a search space of 256.
8queens has 8 queens with a search space of 10^7.
16queens has 16 queens with a search space of 10^19.
32queens has 32 queens with a search space of 10^48.
64queens has 64 queens with a search space of 10^115.
256queens has 256 queens with a search space of 10^616.
```

**n** クイーンは、初心者用のサンプルとして実装されているため、最適化はされていません。それにもかかわらず、クイーンが64個になっても簡単に処理できます。何点か変更を加えると、クイーンが5000個以上になっても簡単に対応できることが立証されています。

### 3.6.1. N クイーンのドメインモデル

この例では、4つのクイーンの問題を解決するドメインモデルを使用します。

- **ドメインモデルの作成**

適切なドメインモデルを使用すると、プランニングの問題をより簡単に理解し、解決することができます。

以下は、**n** クイーンの例のドメインモデルです。

```
public class Column {

    private int index;

    // ... getters and setters
}
```

```

public class Row {

    private int index;

    // ... getters and setters
}

public class Queen {

    private Column column;
    private Row row;

    public int getAscendingDiagonalIndex() {...}
    public int getDescendingDiagonalIndex() {...}

    // ... getters and setters
}

```

- 探索空間の計算

**Queen** インスタンスには **Column** (例: 0 は列 A、1 は列 B) および **Row** (例: 0 は行 0、1 は行 1) が含まれます。

列と行をもとに、昇順の対角線、および降順の対角線を計算することができます。

列と行のインデックスは、チェスボードの左上隅から数えています。

```

public class NQueens {

    private int n;
    private List<Column> columnList;
    private List<Row> rowList;

    private List<Queen> queenList;

    private SimpleScore score;

    // ... getters and setters
}

```

- 解の求め方

1つの **NQueens** インスタンスには **Queen** インスタンスの一覧が含まれています。これが **Solution** 実装として提供され、Solver が解決して読み出します。

たとえば、4 クイーンのサンプルでは、NQueens の **getN()** メソッドが常に 4 を返します。



図3.3 キーーン 4 個の解

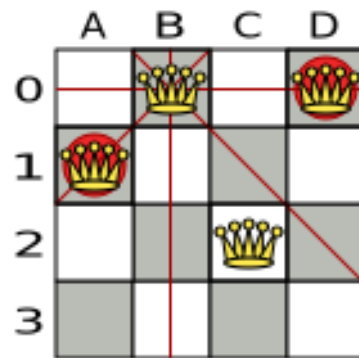


表3.2 ドメインモデルの解の詳細

	columnIndex	rowIndex	ascendingDiagonalIndex (columnIndex + rowIndex)	descendingDiagonalIndex (columnIndex - rowIndex)
A1	0	1	1(**)	-1
B0	1	0(*)	1(**)	1
C2	2	2	4	0
D0	3	0(*)	3	3

(\*) や (\*\*) のように、キーーン 2 つが同じ行、列、対角線を共有する場合は、2 つの駒が互いを取ることができます。

### 3.7. クラウドバランシング

この例に関する詳細は、[Red Hat ビルドの OptaPlanner クイックスタートガイド](#)を参照してください。

### 3.8. 巡回セールスマン (TSP - 巡回セールスマン問題)

都市の一覧をもとに、セールスマンが最短距離で、各都市を 1 度だけ訪問するルートを探します。

この問題は [ウィキペディア](#) に定義されています。これは、計算数学で [最も熱心に研究された問題の 1 つ](#) です。大概は、従業員のシフト勤務など、その他の制約と一緒に計画の問題の一部として使用されます。

#### 問題の規模

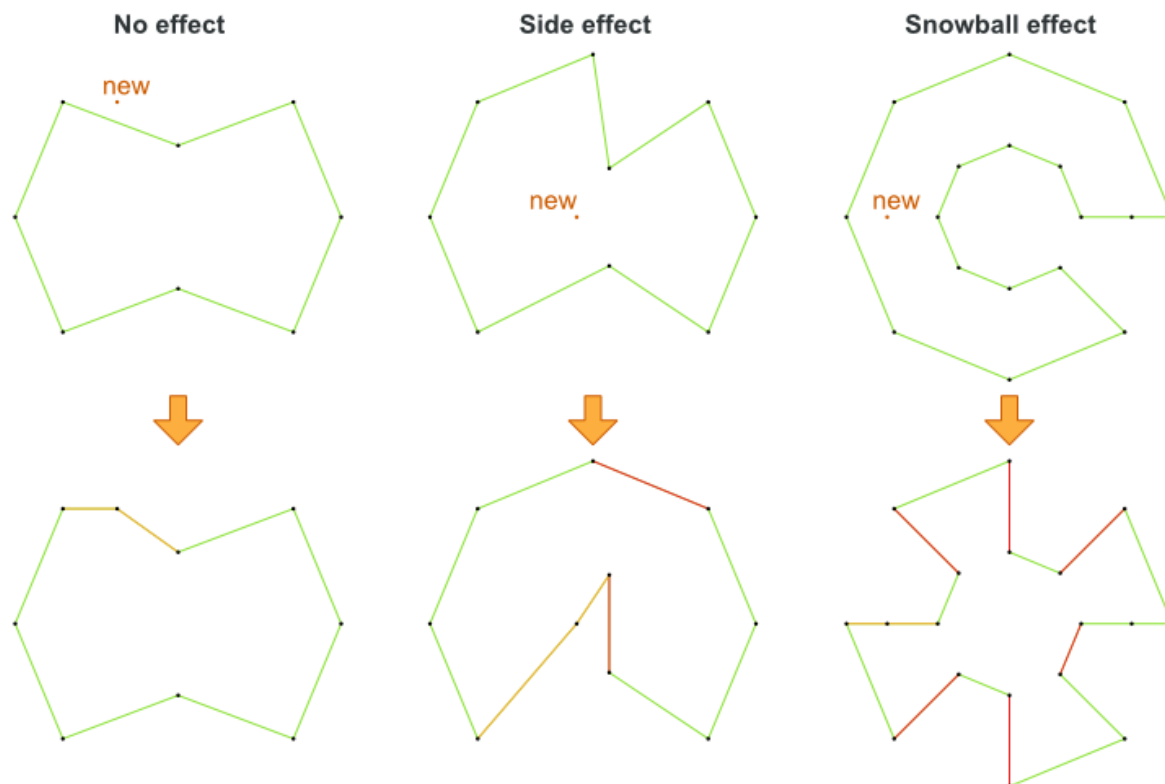
dj38 has 38 cities with a search space of  $10^{43}$ .  
 europe40 has 40 cities with a search space of  $10^{46}$ .  
 st70 has 70 cities with a search space of  $10^{98}$ .  
 pcb442 has 442 cities with a search space of  $10^{976}$ .  
 lu980 has 980 cities with a search space of  $10^{2504}$ .

#### 問題の難易度

TSP の定義は単純ですが、問題の解決は驚くほど難しくなります。これは NP 困難問題と呼ばれ、多くの計画の問題と同様、特定の問題のデータセットに対する最適解は、その問題のデータセットが少しでも変更すると、大幅に変化する可能性があります。

## TSP optimal solution volatility

How much does the optimal solution change if we add 1 new location?



### 3.9. テニスクラブのスケジュール

テニスクラブでは、毎週 4 チームが総あたりで試合をします。4 つの対戦枠を公平にチームに割り当てます。

ハード制約:

- 競合: チームは 1 日に 1 回だけ試合ができる。
- 参加不可: 日程によって参加できないチームがある。

中程度の制約:

- 公平な割り当て: 各チームが試合をする回数を (ほぼ) 同じにする。

ソフト制約:

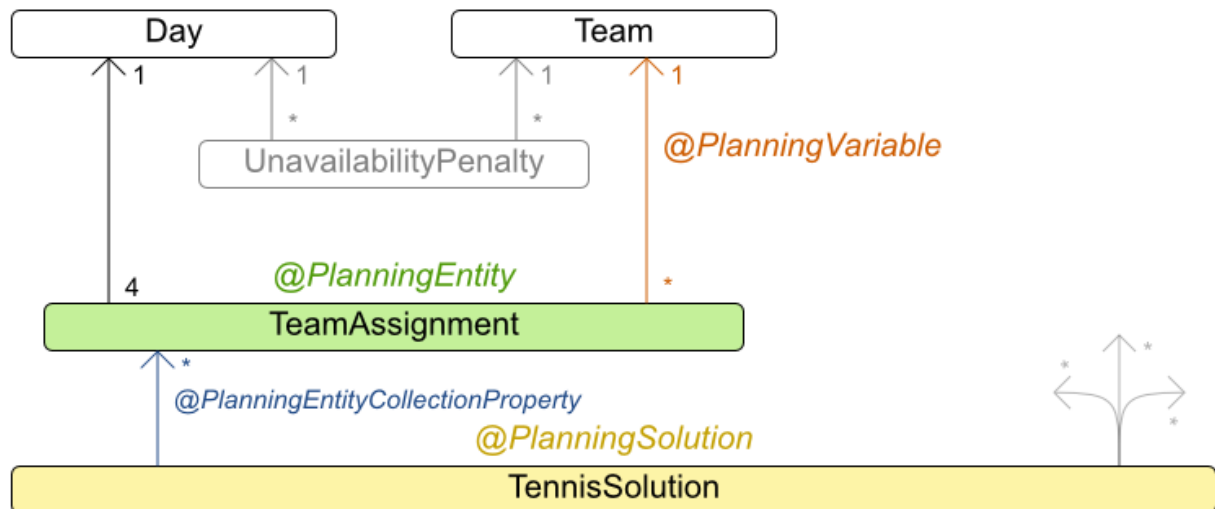
- 均等に対戦: 各チームが、各対戦相手と対戦する回数を同じにする。

#### 問題の規模

munich-7teams has 7 teams, 18 days, 12 unavailabilityPenalties and 72 teamAssignments with a search space of  $10^{60}$ .

図3.4 ドメインモデル

## Tennis class diagram



### 3.10. 会議のスケジュール

各会議に、開始時間と会議室を割り当てます。会議の長さは異なります。

ハード制約:

- 部屋の制約: 2つの会議が、同じ時間に同じ会議室を使用することはできない。
- 必須の出席者: 同じ時間に開催される必須の会議を2つ割り当てることはできない。
- 必要とされる部屋の収容人数: 会議の出席者全員を収容できない部屋では会議を行ってはいけません。
- 同日中に開始して終了: 会議は複数の日にわたってスケジュールされないようにする。

中程度の制約:

- 任意の出席者: 同じ時間に開催される任意の会議を2つ割り当てることはできない。また、任意の会議と必須の会議を同じ時間に割り当てることはできない。

ソフト制約:

- 早い段階でスケジュール: すべての会議をできるだけ早くスケジュールする。
- 会議と会議の間の休憩時間: 会議と会議の間には、最低でも時間枠1つ分、休憩を入れる必要がある。

- 会議の重複: 並行して行われる会議の数を最小限に抑えて、どちらかの会議を選択しなければならない状況をなくす。
- 先に大きい部屋から割り当てる: 参加者が登録していない場合でも、できるだけ多数の参加者を収容するために、大きい部屋が空いている場合にはその部屋から割り当てていく必要がある。
- 部屋の不変性: 会議が連続して行われ、休憩の時間枠が 2 つ分より少ない場合には、会議は同じ部屋で行う方が良い。

## 問題の規模

50meetings-160timegrains-5rooms has 50 meetings, 160 timeGrains and 5 rooms with a search space of  $10^{145}$ .  
 100meetings-320timegrains-5rooms has 100 meetings, 320 timeGrains and 5 rooms with a search space of  $10^{320}$ .  
 200meetings-640timegrains-5rooms has 200 meetings, 640 timeGrains and 5 rooms with a search space of  $10^{701}$ .  
 400meetings-1280timegrains-5rooms has 400 meetings, 1280 timeGrains and 5 rooms with a search space of  $10^{1522}$ .  
 800meetings-2560timegrains-5rooms has 800 meetings, 2560 timeGrains and 5 rooms with a search space of  $10^{3285}$ .

## 3.11. コースの時間割 (ITC 2007 TRACK 3 - カリキュラムのスケジュール)

各授業を、時間枠および講義室に割り当ててスケジュールを組みます。

ハード制約:

- 講師の制約: 各講師は、同じ時間に授業を 2 つ受け持つことはできない。
- カリキュラムの制約: カリキュラムには、2 つの授業を同じ時間に設定することはできない。
- 部屋の占有: 同じ時間の同じ講義室に、2 つの授業を割り当てることはできない。
- 利用不可の時間 (データセットごとに指定): 授業には割り当てられない時間がある。

ソフト制約:

- 講義室の収容人数: 講義室の収容人数は、その授業を受ける学生の数よりも多くなければならない。
- 最小限の就業日数: 同じコースの授業の開講期間は、最短になるようにする。
- カリキュラムの緊密さ: 同じカリキュラムに含まれる授業は、時間帯を近く (連続した時間に) 設定する。
- 講義室の不変性: 同じコースの授業は同じ講義室を割り当てる必要がある。

この問題は、[International Timetabling Competition 2007 track 3](#) で定義されています。

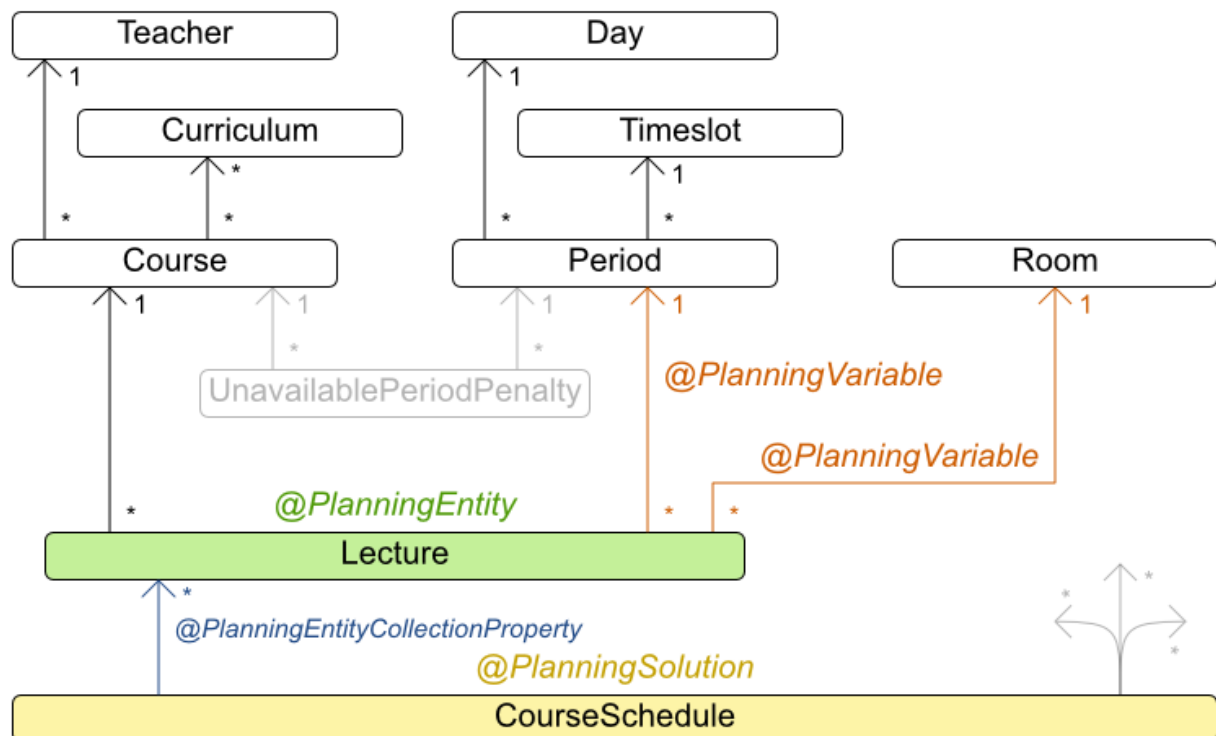
## 問題の規模

comp01 has 24 teachers, 14 curricula, 30 courses, 160 lectures, 30 periods, 6 rooms and 53 unavailable period constraints with a search space of  $10^{360}$ .  
 comp02 has 71 teachers, 70 curricula, 82 courses, 283 lectures, 25 periods, 16 rooms and 513 unavailable period constraints with a search space of  $10^{736}$ .

comp03 has 61 teachers, 68 curricula, 72 courses, 251 lectures, 25 periods, 16 rooms and 382 unavailable period constraints with a search space of  $10^{653}$ .  
 comp04 has 70 teachers, 57 curricula, 79 courses, 286 lectures, 25 periods, 18 rooms and 396 unavailable period constraints with a search space of  $10^{758}$ .  
 comp05 has 47 teachers, 139 curricula, 54 courses, 152 lectures, 36 periods, 9 rooms and 771 unavailable period constraints with a search space of  $10^{381}$ .  
 comp06 has 87 teachers, 70 curricula, 108 courses, 361 lectures, 25 periods, 18 rooms and 632 unavailable period constraints with a search space of  $10^{957}$ .  
 comp07 has 99 teachers, 77 curricula, 131 courses, 434 lectures, 25 periods, 20 rooms and 667 unavailable period constraints with a search space of  $10^{1171}$ .  
 comp08 has 76 teachers, 61 curricula, 86 courses, 324 lectures, 25 periods, 18 rooms and 478 unavailable period constraints with a search space of  $10^{859}$ .  
 comp09 has 68 teachers, 75 curricula, 76 courses, 279 lectures, 25 periods, 18 rooms and 405 unavailable period constraints with a search space of  $10^{740}$ .  
 comp10 has 88 teachers, 67 curricula, 115 courses, 370 lectures, 25 periods, 18 rooms and 694 unavailable period constraints with a search space of  $10^{981}$ .  
 comp11 has 24 teachers, 13 curricula, 30 courses, 162 lectures, 45 periods, 5 rooms and 94 unavailable period constraints with a search space of  $10^{381}$ .  
 comp12 has 74 teachers, 150 curricula, 88 courses, 218 lectures, 36 periods, 11 rooms and 1368 unavailable period constraints with a search space of  $10^{566}$ .  
 comp13 has 77 teachers, 66 curricula, 82 courses, 308 lectures, 25 periods, 19 rooms and 468 unavailable period constraints with a search space of  $10^{824}$ .  
 comp14 has 68 teachers, 60 curricula, 85 courses, 275 lectures, 25 periods, 17 rooms and 486 unavailable period constraints with a search space of  $10^{722}$ .

図3.5 ドメインモデル

## Curriculum course class diagram



### 3.12. マシンの再割当て (GOOGLE ROADEF 2012)

各プロセスをマシンに割り当てます。全プロセスには、すでに元の (最適化されていない) 割り当てがあります。プロセスにはそれぞれ、各リソース (CPU、メモリーなど) が一定量必要です。これは、クラウドのバランスの例の応用です。

ハード制約:

- 最大容量: マシンに割り当てる各リソースはこの量を超えてはいけません。
- 競合: 同じサービスのプロセスは別のマシンで実行する必要があります。
- 分散: 同じサービスのプロセスは複数の場所に分散させる必要があります。
- 依存関係: 他のサービスに依存するサービスのプロセスは、そのサービスの近くで実行する必要があります。
- 一時的な使用: リソースによっては一時的なものがあり、元のマシンと、新たに割り当てられたマシンの両方の最大容量にカウントされる。

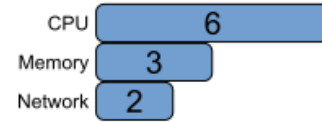
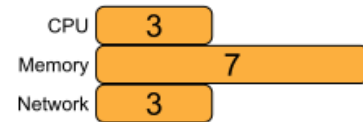
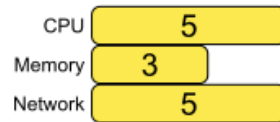
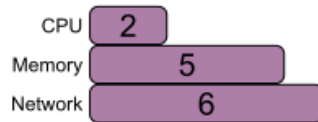
ソフト制約:

- 負荷: 各マシンの各リソースの安全容量を超えてはいけません。
- 負荷分散: 各マシンで利用可能なリソースを分散させて、今後の割り当てに対応できるように容量を空ける。
- プロセスの移動コスト: プロセスには移動コストが発生する。
- サービスの移動コスト: サービスには移動コストが発生する。
- 機械の移動コスト: マシン A からマシン B にプロセスを移動すると、A から B に固有の移動コストが別途発生する。

この問題は [the Google ROADEF/EURO Challenge 2012](#) で定義されています。

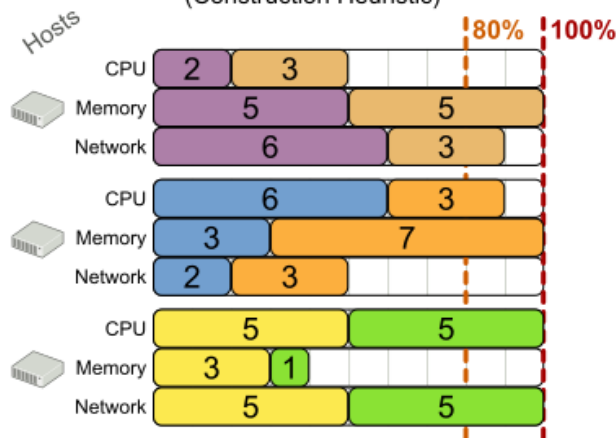
# Cloud optimization is like Tetris

Processes



## Traditional algorithm

(Construction Heuristic)



## OptaPlanner

(Construction Heuristic + Local Search)

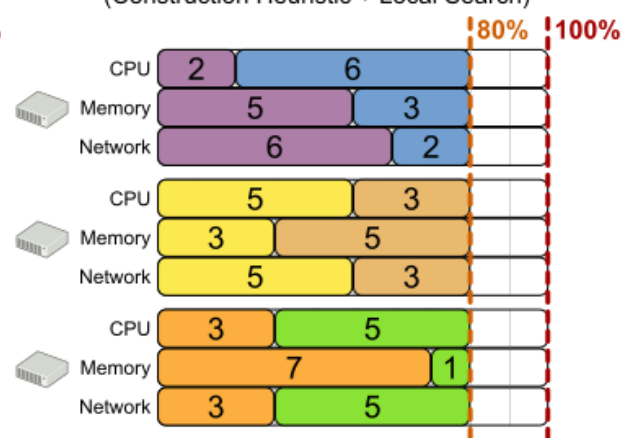
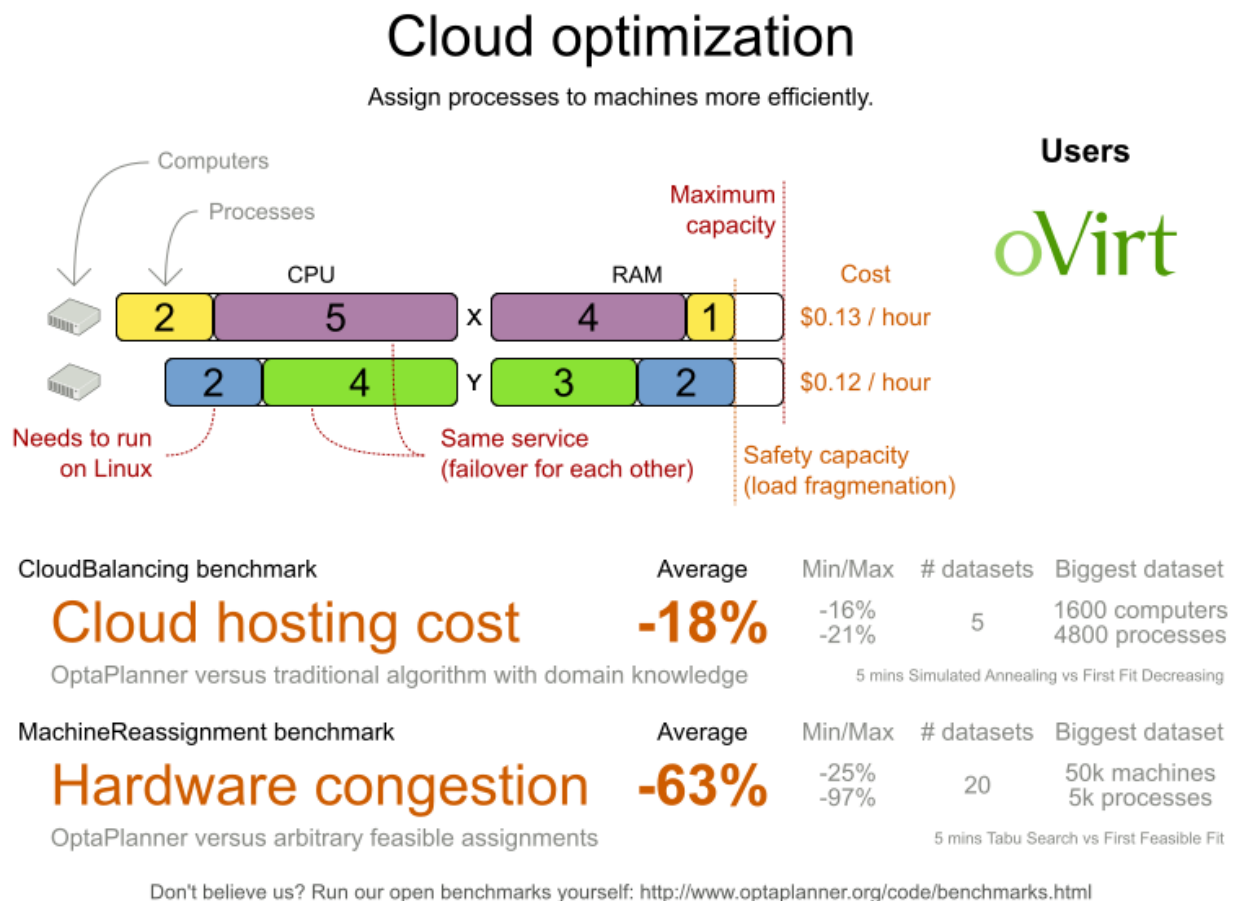


図3.6 価値提案



## 問題の規模

model\_a1\_1 has 2 resources, 1 neighborhoods, 4 locations, 4 machines, 79 services, 100 processes and 1 balancePenalties with a search space of  $10^{60}$ .

model\_a1\_2 has 4 resources, 2 neighborhoods, 4 locations, 100 machines, 980 services, 1000 processes and 0 balancePenalties with a search space of  $10^{2000}$ .

model\_a1\_3 has 3 resources, 5 neighborhoods, 25 locations, 100 machines, 216 services, 1000 processes and 0 balancePenalties with a search space of  $10^{2000}$ .

model\_a1\_4 has 3 resources, 50 neighborhoods, 50 locations, 50 machines, 142 services, 1000 processes and 1 balancePenalties with a search space of  $10^{1698}$ .

model\_a1\_5 has 4 resources, 2 neighborhoods, 4 locations, 12 machines, 981 services, 1000 processes and 1 balancePenalties with a search space of  $10^{1079}$ .

model\_a2\_1 has 3 resources, 1 neighborhoods, 1 locations, 100 machines, 1000 services, 1000 processes and 0 balancePenalties with a search space of  $10^{2000}$ .

model\_a2\_2 has 12 resources, 5 neighborhoods, 25 locations, 100 machines, 170 services, 1000 processes and 0 balancePenalties with a search space of  $10^{2000}$ .

model\_a2\_3 has 12 resources, 5 neighborhoods, 25 locations, 100 machines, 129 services, 1000 processes and 0 balancePenalties with a search space of  $10^{2000}$ .

model\_a2\_4 has 12 resources, 5 neighborhoods, 25 locations, 50 machines, 180 services, 1000 processes and 1 balancePenalties with a search space of  $10^{1698}$ .

model\_a2\_5 has 12 resources, 5 neighborhoods, 25 locations, 50 machines, 153 services, 1000 processes and 0 balancePenalties with a search space of  $10^{1698}$ .

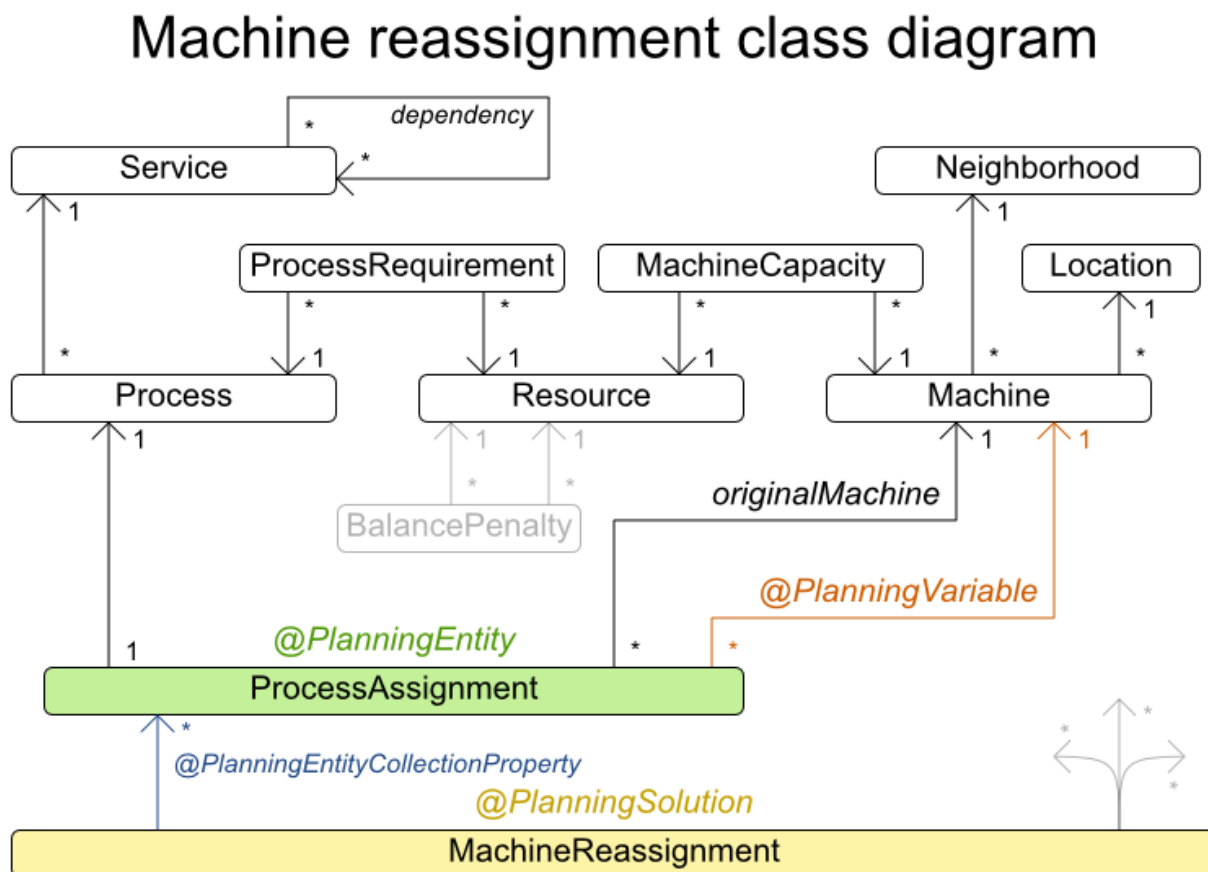
model\_b\_1 has 12 resources, 5 neighborhoods, 10 locations, 100 machines, 2512 services, 5000 processes and 0 balancePenalties with a search space of  $10^{10000}$ .

model\_b\_2 has 12 resources, 5 neighborhoods, 10 locations, 100 machines, 2462 services, 5000 processes and 1 balancePenalties with a search space of  $10^{10000}$ .



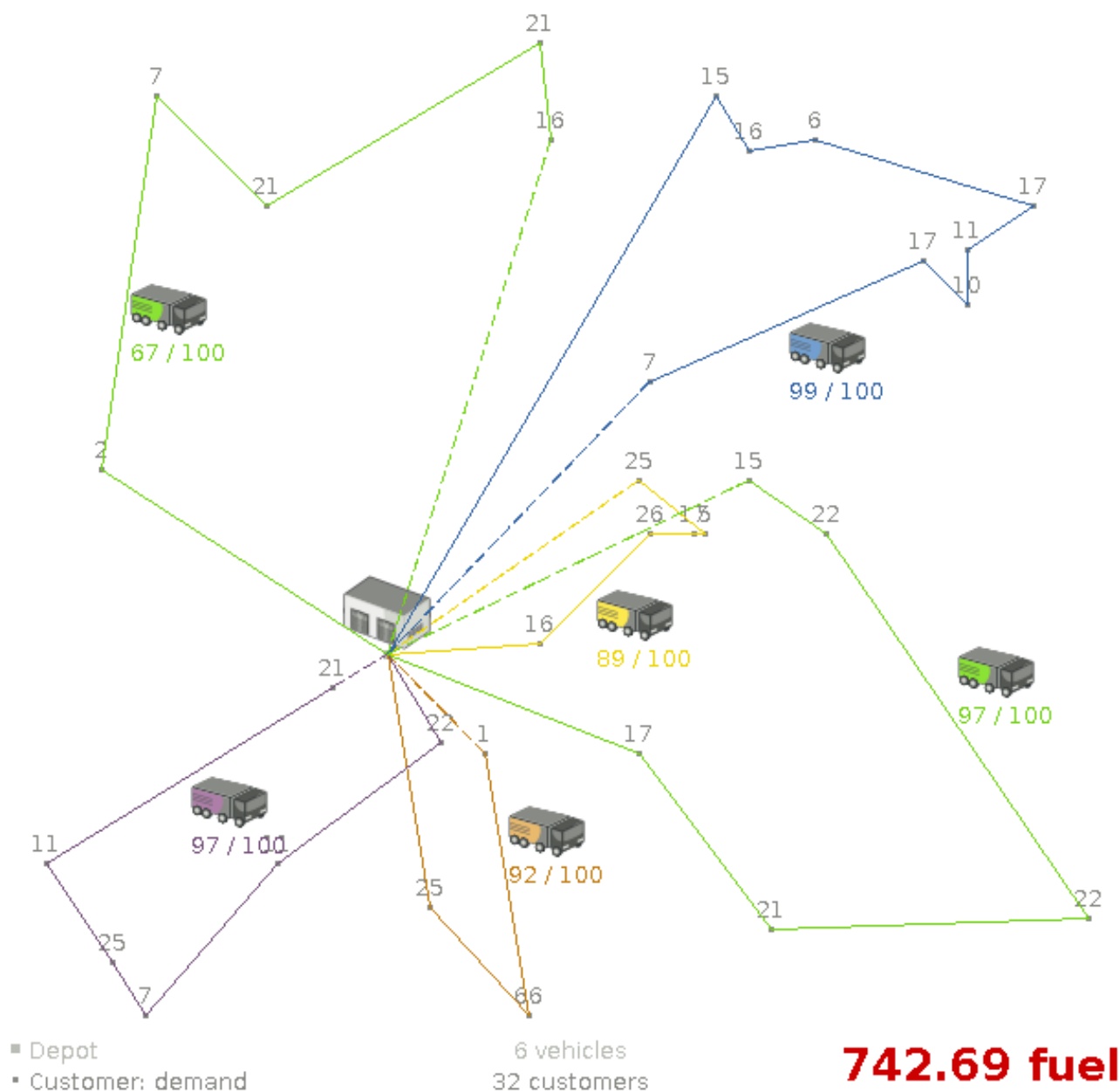
model\_b\_3 has 6 resources, 5 neighborhoods, 10 locations, 100 machines, 15025 services, 20000 processes and 0 balancePenalties with a search space of  $10^{40000}$ .  
 model\_b\_4 has 6 resources, 5 neighborhoods, 50 locations, 500 machines, 1732 services, 20000 processes and 1 balancePenalties with a search space of  $10^{53979}$ .  
 model\_b\_5 has 6 resources, 5 neighborhoods, 10 locations, 100 machines, 35082 services, 40000 processes and 0 balancePenalties with a search space of  $10^{80000}$ .  
 model\_b\_6 has 6 resources, 5 neighborhoods, 50 locations, 200 machines, 14680 services, 40000 processes and 1 balancePenalties with a search space of  $10^{92041}$ .  
 model\_b\_7 has 6 resources, 5 neighborhoods, 50 locations, 4000 machines, 15050 services, 40000 processes and 1 balancePenalties with a search space of  $10^{144082}$ .  
 model\_b\_8 has 3 resources, 5 neighborhoods, 10 locations, 100 machines, 45030 services, 50000 processes and 0 balancePenalties with a search space of  $10^{100000}$ .  
 model\_b\_9 has 3 resources, 5 neighborhoods, 100 locations, 1000 machines, 4609 services, 50000 processes and 1 balancePenalties with a search space of  $10^{150000}$ .  
 model\_b\_10 has 3 resources, 5 neighborhoods, 100 locations, 5000 machines, 4896 services, 50000 processes and 1 balancePenalties with a search space of  $10^{184948}$ .

図3.7 ドメインモデル



### 3.13. 配送経路

複数の車両を使用して、各顧客の品物を回収し、倉庫まで運びます。1つの車両で複数の顧客から品物を回収することはできますが、収容できる容量には限りがあります。



基本例 (CVRP) のほかに、時間枠の設定が加わった例 (CVRPTW) もあります。

ハード制約:

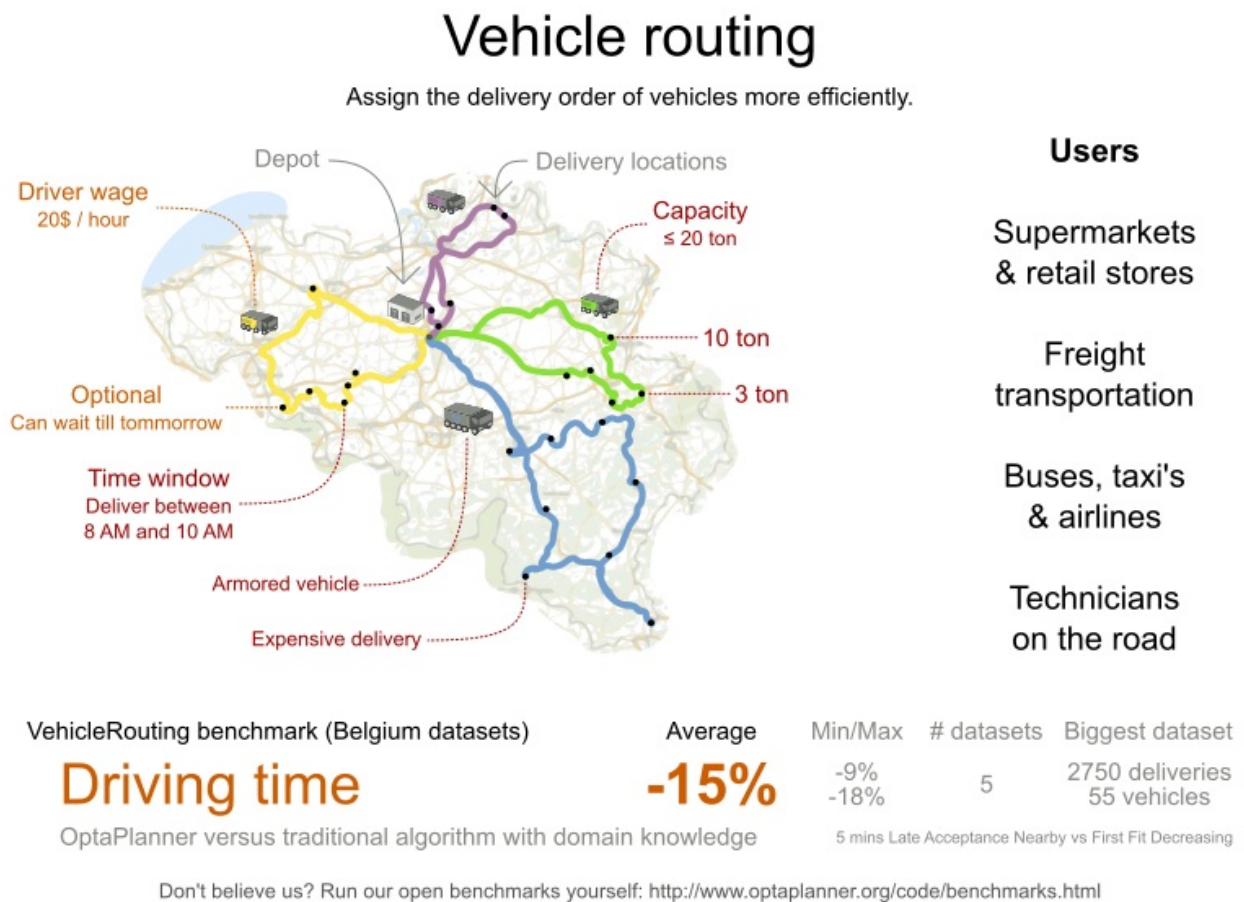
- 車両の容量: 車両は、車載容量を超えて品物を運ぶことができない。
- 時間枠 (CVRPTW のみ):
  - 移動時間: 別の場所に移動する場合には時間がかかる。
  - 顧客対応の時間: 車両は顧客に対応している時間、顧客先にとどまる必要がある。
  - 顧客の準備が整う時間: 顧客の準備が整う前に車両が到着する可能性があるが、準備ができるまで待機してから顧客に対応する必要がある。
  - 顧客が設定した締め切り時間: 車両は、顧客が設定した締め切り時間までに到着する必要がある。

ソフト制約:

- 合計距離: 車両が移動する合計距離 (ガソリンの消費量) を最小限に抑える。

CVRP (Capacitated Vehicle Routing Problem) と CVRPTW (Capacitated Vehicle Routing Problem with Time Window) は、VRP Web で定義されています。

図3.8 価値提案



## 問題の規模

CVRP インスタンス (時間枠なし):

belgium-n50-k10	has 1 depots, 10 vehicles and 49 customers with a search space of $10^{74}$ .
belgium-n100-k10	has 1 depots, 10 vehicles and 99 customers with a search space of $10^{170}$ .
belgium-n500-k20	has 1 depots, 20 vehicles and 499 customers with a search space of $10^{1168}$ .
belgium-n1000-k20	has 1 depots, 20 vehicles and 999 customers with a search space of $10^{2607}$ .
belgium-n2750-k55	has 1 depots, 55 vehicles and 2749 customers with a search space of $10^{8380}$ .
belgium-road-km-n50-k10	has 1 depots, 10 vehicles and 49 customers with a search space of $10^{74}$ .
belgium-road-km-n100-k10	has 1 depots, 10 vehicles and 99 customers with a search space of $10^{170}$ .
belgium-road-km-n500-k20	has 1 depots, 20 vehicles and 499 customers with a search space of $10^{1168}$ .
belgium-road-km-n1000-k20	has 1 depots, 20 vehicles and 999 customers with a search space of $10^{2607}$ .
belgium-road-km-n2750-k55	has 1 depots, 55 vehicles and 2749 customers with a search space of

10<sup>8380</sup>.

belgium-road-time-n50-k10 has 1 depots, 10 vehicles and 49 customers with a search space of 10<sup>74</sup>.

belgium-road-time-n100-k10 has 1 depots, 10 vehicles and 99 customers with a search space of 10<sup>170</sup>.

belgium-road-time-n500-k20 has 1 depots, 20 vehicles and 499 customers with a search space of 10<sup>1168</sup>.

belgium-road-time-n1000-k20 has 1 depots, 20 vehicles and 999 customers with a search space of 10<sup>2607</sup>.

belgium-road-time-n2750-k55 has 1 depots, 55 vehicles and 2749 customers with a search space of 10<sup>8380</sup>.

belgium-d2-n50-k10 has 2 depots, 10 vehicles and 48 customers with a search space of 10<sup>74</sup>.

belgium-d3-n100-k10 has 3 depots, 10 vehicles and 97 customers with a search space of 10<sup>170</sup>.

belgium-d5-n500-k20 has 5 depots, 20 vehicles and 495 customers with a search space of 10<sup>1168</sup>.

belgium-d8-n1000-k20 has 8 depots, 20 vehicles and 992 customers with a search space of 10<sup>2607</sup>.

belgium-d10-n2750-k55 has 10 depots, 55 vehicles and 2740 customers with a search space of 10<sup>8380</sup>.

A-n32-k5 has 1 depots, 5 vehicles and 31 customers with a search space of 10<sup>40</sup>.

A-n33-k5 has 1 depots, 5 vehicles and 32 customers with a search space of 10<sup>41</sup>.

A-n33-k6 has 1 depots, 6 vehicles and 32 customers with a search space of 10<sup>42</sup>.

A-n34-k5 has 1 depots, 5 vehicles and 33 customers with a search space of 10<sup>43</sup>.

A-n36-k5 has 1 depots, 5 vehicles and 35 customers with a search space of 10<sup>46</sup>.

A-n37-k5 has 1 depots, 5 vehicles and 36 customers with a search space of 10<sup>48</sup>.

A-n37-k6 has 1 depots, 6 vehicles and 36 customers with a search space of 10<sup>49</sup>.

A-n38-k5 has 1 depots, 5 vehicles and 37 customers with a search space of 10<sup>49</sup>.

A-n39-k5 has 1 depots, 5 vehicles and 38 customers with a search space of 10<sup>51</sup>.

A-n39-k6 has 1 depots, 6 vehicles and 38 customers with a search space of 10<sup>52</sup>.

A-n44-k7 has 1 depots, 7 vehicles and 43 customers with a search space of 10<sup>61</sup>.

A-n45-k6 has 1 depots, 6 vehicles and 44 customers with a search space of 10<sup>62</sup>.

A-n45-k7 has 1 depots, 7 vehicles and 44 customers with a search space of 10<sup>63</sup>.

A-n46-k7 has 1 depots, 7 vehicles and 45 customers with a search space of 10<sup>65</sup>.

A-n48-k7 has 1 depots, 7 vehicles and 47 customers with a search space of 10<sup>68</sup>.

A-n53-k7 has 1 depots, 7 vehicles and 52 customers with a search space of 10<sup>77</sup>.

A-n54-k7 has 1 depots, 7 vehicles and 53 customers with a search space of 10<sup>79</sup>.

A-n55-k9 has 1 depots, 9 vehicles and 54 customers with a search space of 10<sup>82</sup>.

A-n60-k9 has 1 depots, 9 vehicles and 59 customers with a search space of 10<sup>91</sup>.

A-n61-k9 has 1 depots, 9 vehicles and 60 customers with a search space of 10<sup>93</sup>.

A-n62-k8 has 1 depots, 8 vehicles and 61 customers with a search space of 10<sup>94</sup>.

A-n63-k9 has 1 depots, 9 vehicles and 62 customers with a search space of 10<sup>97</sup>.

A-n63-k10 has 1 depots, 10 vehicles and 62 customers with a search space of 10<sup>98</sup>.

A-n64-k9 has 1 depots, 9 vehicles and 63 customers with a search space of 10<sup>99</sup>.

A-n65-k9 has 1 depots, 9 vehicles and 64 customers with a search space of 10<sup>101</sup>.

A-n69-k9 has 1 depots, 9 vehicles and 68 customers with a search space of 10<sup>108</sup>.

A-n80-k10 has 1 depots, 10 vehicles and 79 customers with a search space of 10<sup>130</sup>.

F-n45-k4 has 1 depots, 4 vehicles and 44 customers with a search space of 10<sup>60</sup>.

F-n72-k4 has 1 depots, 4 vehicles and 71 customers with a search space of 10<sup>108</sup>.

F-n135-k7 has 1 depots, 7 vehicles and 134 customers with a search space of 10<sup>240</sup>.

CVRPTW インスタンス (時間枠あり):

belgium-tw-d2-n50-k10 has 2 depots, 10 vehicles and 48 customers with a search space of

10<sup>4</sup>.

belgium-tw-d3-n100-k10 has 3 depots, 10 vehicles and 97 customers with a search space of 10<sup>170</sup>.

belgium-tw-d5-n500-k20 has 5 depots, 20 vehicles and 495 customers with a search space of 10<sup>1168</sup>.

belgium-tw-d8-n1000-k20 has 8 depots, 20 vehicles and 992 customers with a search space of 10<sup>2607</sup>.

belgium-tw-d10-n2750-k55 has 10 depots, 55 vehicles and 2740 customers with a search space of 10<sup>8380</sup>.

belgium-tw-n50-k10 has 1 depots, 10 vehicles and 49 customers with a search space of 10<sup>74</sup>.

belgium-tw-n100-k10 has 1 depots, 10 vehicles and 99 customers with a search space of 10<sup>170</sup>.

belgium-tw-n500-k20 has 1 depots, 20 vehicles and 499 customers with a search space of 10<sup>1168</sup>.

belgium-tw-n1000-k20 has 1 depots, 20 vehicles and 999 customers with a search space of 10<sup>2607</sup>.

belgium-tw-n2750-k55 has 1 depots, 55 vehicles and 2749 customers with a search space of 10<sup>8380</sup>.

Solomon\_025\_C101 has 1 depots, 25 vehicles and 25 customers with a search space of 10<sup>40</sup>.

Solomon\_025\_C201 has 1 depots, 25 vehicles and 25 customers with a search space of 10<sup>40</sup>.

Solomon\_025\_R101 has 1 depots, 25 vehicles and 25 customers with a search space of 10<sup>40</sup>.

Solomon\_025\_R201 has 1 depots, 25 vehicles and 25 customers with a search space of 10<sup>40</sup>.

Solomon\_025\_RC101 has 1 depots, 25 vehicles and 25 customers with a search space of 10<sup>40</sup>.

Solomon\_025\_RC201 has 1 depots, 25 vehicles and 25 customers with a search space of 10<sup>40</sup>.

Solomon\_100\_C101 has 1 depots, 25 vehicles and 100 customers with a search space of 10<sup>185</sup>.

Solomon\_100\_C201 has 1 depots, 25 vehicles and 100 customers with a search space of 10<sup>185</sup>.

Solomon\_100\_R101 has 1 depots, 25 vehicles and 100 customers with a search space of 10<sup>185</sup>.

Solomon\_100\_R201 has 1 depots, 25 vehicles and 100 customers with a search space of 10<sup>185</sup>.

Solomon\_100\_RC101 has 1 depots, 25 vehicles and 100 customers with a search space of 10<sup>185</sup>.

Solomon\_100\_RC201 has 1 depots, 25 vehicles and 100 customers with a search space of 10<sup>185</sup>.

Homberger\_0200\_C1\_2\_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10<sup>429</sup>.

Homberger\_0200\_C2\_2\_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10<sup>429</sup>.

Homberger\_0200\_R1\_2\_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10<sup>429</sup>.

Homberger\_0200\_R2\_2\_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10<sup>429</sup>.

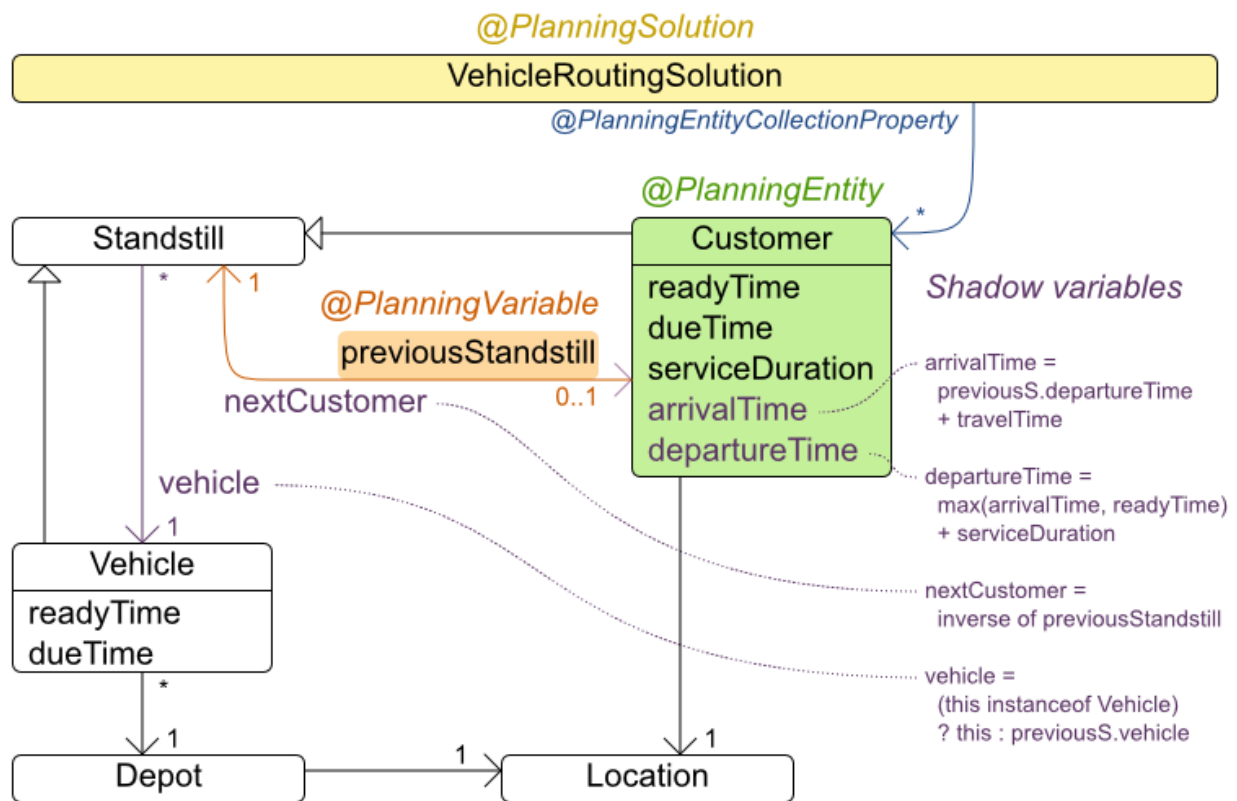
Homberger\_0200\_RC1\_2\_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10<sup>429</sup>.

Homberger\_0200\_RC2\_2\_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10<sup>429</sup>.

Homberger\_0400\_C1\_4\_1 has 1 depots, 100 vehicles and 400 customers with a search space of  $10^{978}$ .  
 Homberger\_0400\_C2\_4\_1 has 1 depots, 100 vehicles and 400 customers with a search space of  $10^{978}$ .  
 Homberger\_0400\_R1\_4\_1 has 1 depots, 100 vehicles and 400 customers with a search space of  $10^{978}$ .  
 Homberger\_0400\_R2\_4\_1 has 1 depots, 100 vehicles and 400 customers with a search space of  $10^{978}$ .  
 Homberger\_0400\_RC1\_4\_1 has 1 depots, 100 vehicles and 400 customers with a search space of  $10^{978}$ .  
 Homberger\_0400\_RC2\_4\_1 has 1 depots, 100 vehicles and 400 customers with a search space of  $10^{978}$ .  
 Homberger\_0600\_C1\_6\_1 has 1 depots, 150 vehicles and 600 customers with a search space of  $10^{1571}$ .  
 Homberger\_0600\_C2\_6\_1 has 1 depots, 150 vehicles and 600 customers with a search space of  $10^{1571}$ .  
 Homberger\_0600\_R1\_6\_1 has 1 depots, 150 vehicles and 600 customers with a search space of  $10^{1571}$ .  
 Homberger\_0600\_R2\_6\_1 has 1 depots, 150 vehicles and 600 customers with a search space of  $10^{1571}$ .  
 Homberger\_0600\_RC1\_6\_1 has 1 depots, 150 vehicles and 600 customers with a search space of  $10^{1571}$ .  
 Homberger\_0600\_RC2\_6\_1 has 1 depots, 150 vehicles and 600 customers with a search space of  $10^{1571}$ .  
 Homberger\_0800\_C1\_8\_1 has 1 depots, 200 vehicles and 800 customers with a search space of  $10^{2195}$ .  
 Homberger\_0800\_C2\_8\_1 has 1 depots, 200 vehicles and 800 customers with a search space of  $10^{2195}$ .  
 Homberger\_0800\_R1\_8\_1 has 1 depots, 200 vehicles and 800 customers with a search space of  $10^{2195}$ .  
 Homberger\_0800\_R2\_8\_1 has 1 depots, 200 vehicles and 800 customers with a search space of  $10^{2195}$ .  
 Homberger\_0800\_RC1\_8\_1 has 1 depots, 200 vehicles and 800 customers with a search space of  $10^{2195}$ .  
 Homberger\_0800\_RC2\_8\_1 has 1 depots, 200 vehicles and 800 customers with a search space of  $10^{2195}$ .  
 Homberger\_1000\_C110\_1 has 1 depots, 250 vehicles and 1000 customers with a search space of  $10^{2840}$ .  
 Homberger\_1000\_C210\_1 has 1 depots, 250 vehicles and 1000 customers with a search space of  $10^{2840}$ .  
 Homberger\_1000\_R110\_1 has 1 depots, 250 vehicles and 1000 customers with a search space of  $10^{2840}$ .  
 Homberger\_1000\_R210\_1 has 1 depots, 250 vehicles and 1000 customers with a search space of  $10^{2840}$ .  
 Homberger\_1000\_RC110\_1 has 1 depots, 250 vehicles and 1000 customers with a search space of  $10^{2840}$ .  
 Homberger\_1000\_RC210\_1 has 1 depots, 250 vehicles and 1000 customers with a search space of  $10^{2840}$ .

### 3.13.1. 配送経路のドメインモデル

# Vehicle routing class diagram



時間枠ありの配送経路のドメインモデルでは、シャドウ変数の機能を多用します。こうすることで、**arrivalTime** や **departureTime** などのプロパティーがドメインモデルで直接利用できるため、制約をより自然に表現できます。

## 直線距離ではなく道路の距離

車は、直線距離を移動するのではなく、道路や高速道路を使用する必要があります。ビジネスの観点からすると、これは非常に重要です。

# Vehicle routing distance type

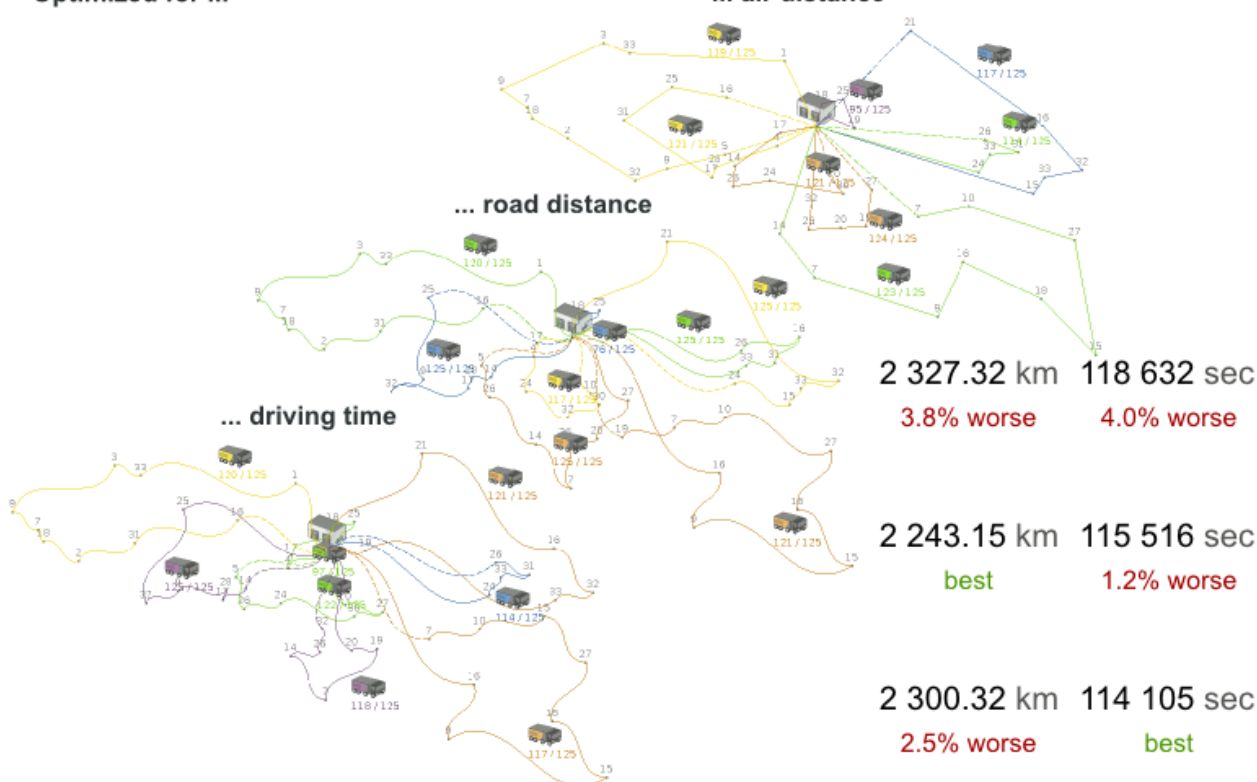
Can we optimize for air distances, when we need road distances or driving times?

Optimized for ...

... air distance

... road distance

... driving time

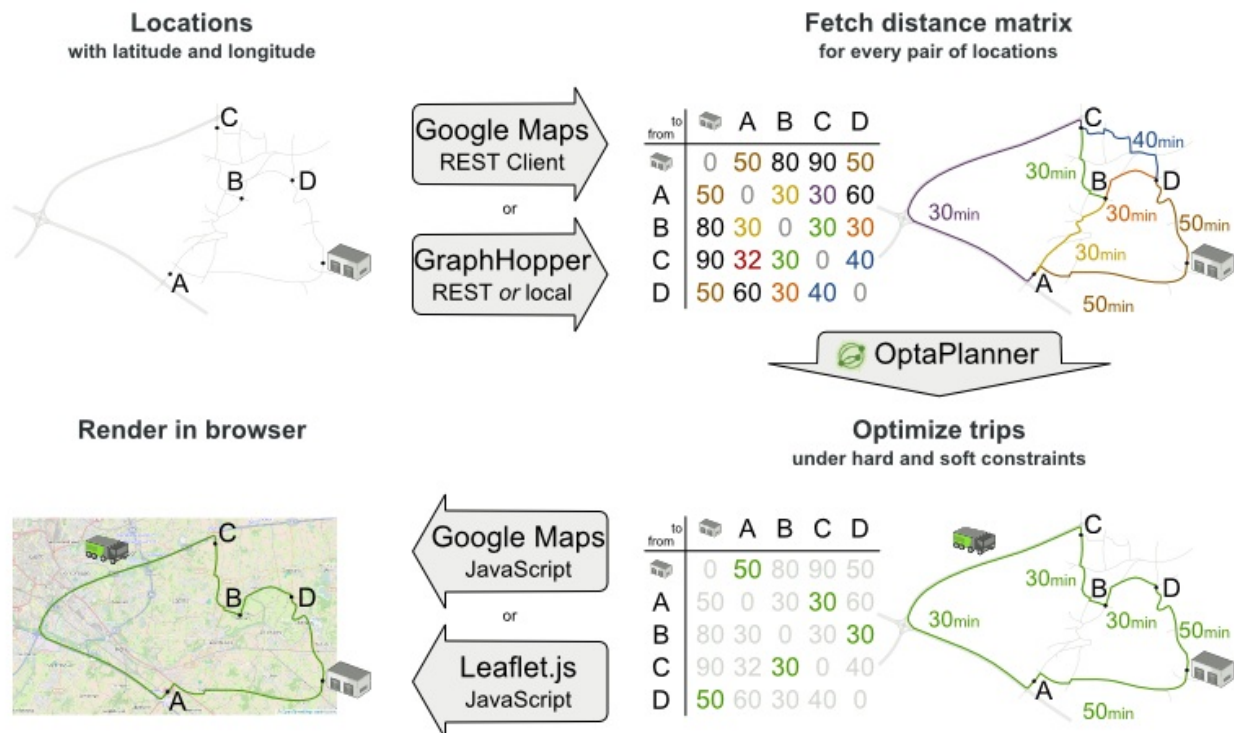


最適化アルゴリズムでは、2 点の距離を検索できている (できれば、事前に計算されている) 場合には、これは特に重要ではありません。道路費は距離である必要はありません。また、移動時間、フラッシュコスト、または加重関数を使用することもできます。GraphHopper (埋め込み可能なオフライン Java エンジン)、Open MapQuest (web サービス)、Google Maps Client API (web サービス) など、移動コストを事前に計算する技術があります。



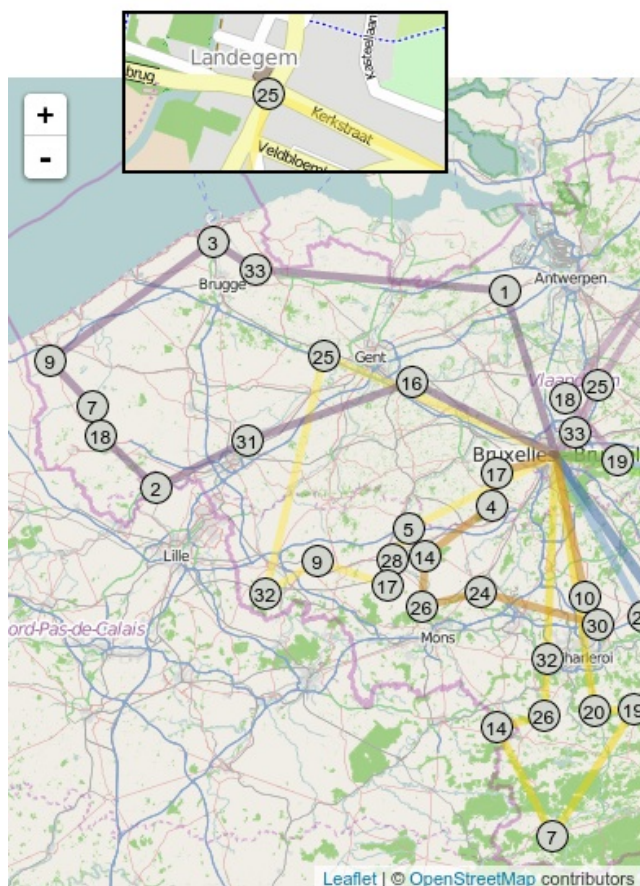
# Integration with real maps

Google Maps or GraphHopper (OpenStreetMap) calculate distances, OptaPlanner optimizes the trips.

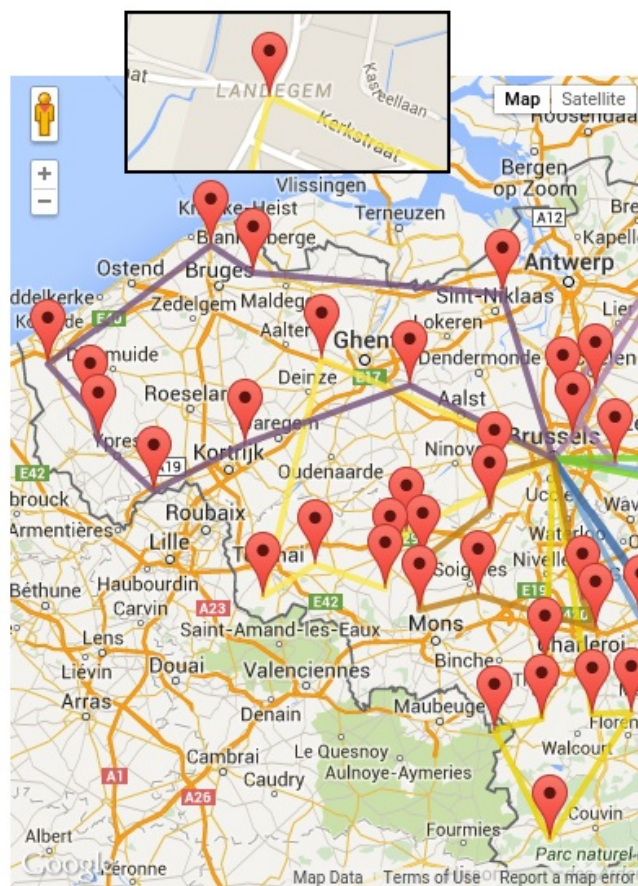


また、[Leaflet](#) や [Google Maps for developers](#) など、レンダリングする技術も複数あります。

## Leaflet.js



## Google Maps



GraphHopper または Google Map Directions を使用して実際の経路をレンダリングすることも可能ですが、高速道路で経路が重なるため、停止する順番を確認するのが困難になります。

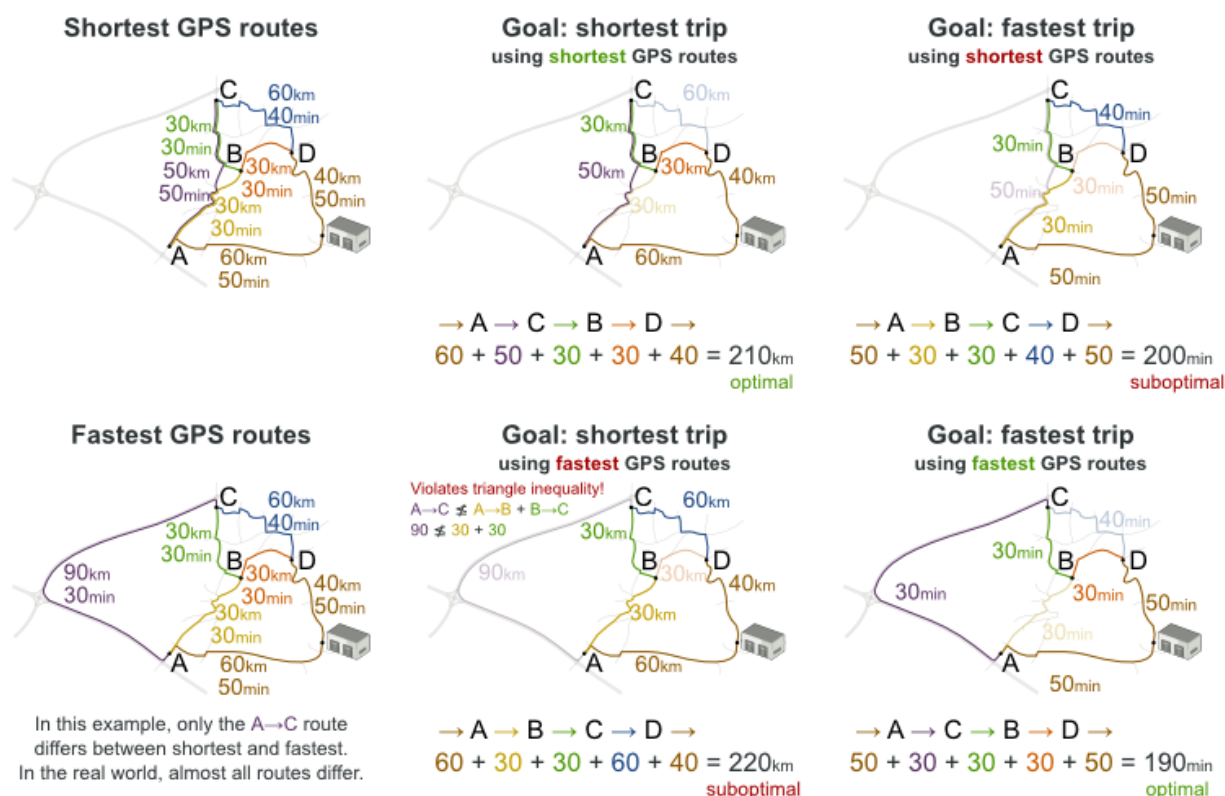




2 点間の移動費は、OptaPlanner で使用されるのと同じ最適化条件を使用する点に注意してください。たとえば、GraphHopper はデフォルトで、最短ではなく、最速の経路を返します。最速の GPS 経路の km (またはマイル) の距離を使用して、OptaPlanner で最短の移動を最適化しないようにしてください。以下のように、準最適解が導き出される可能性があります。

# Road distance triangle inequality

Routes and trips must optimize the same property to avoid suboptimal solutions.



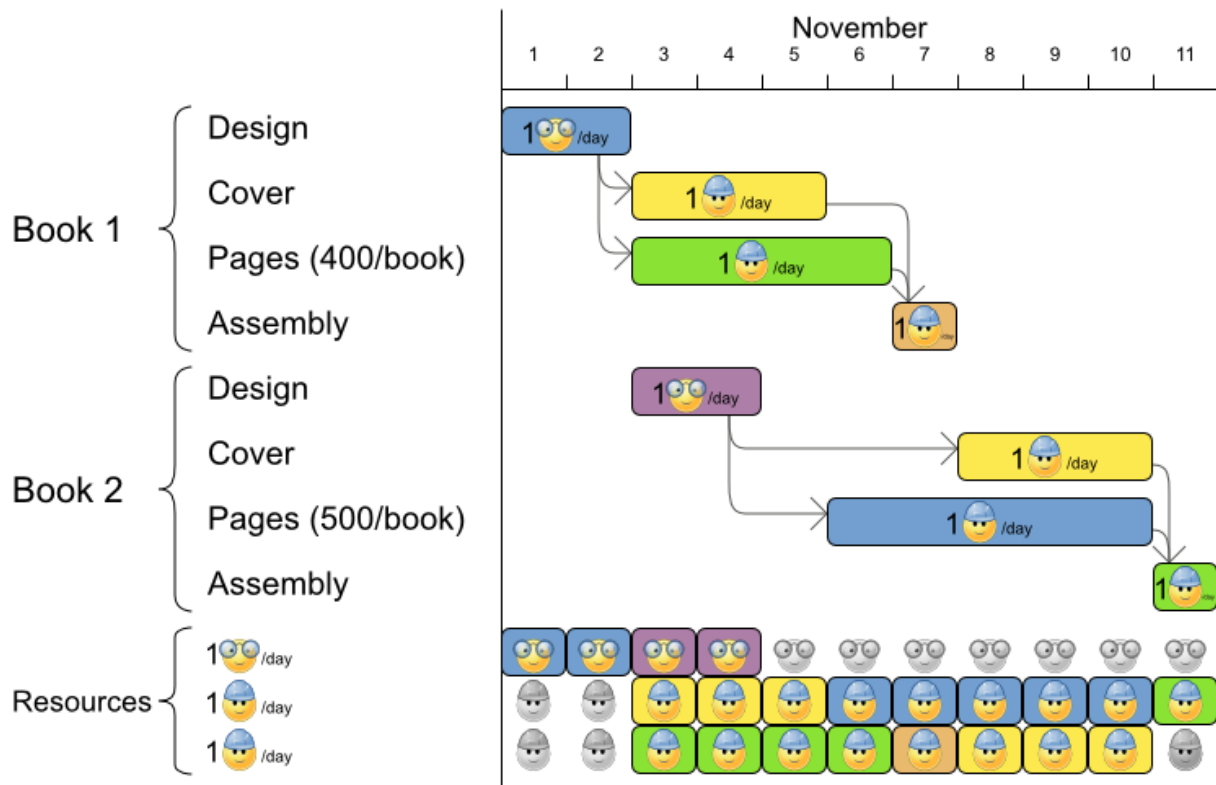
一般的な考え方とは異なり、多くのユーザーは最短の経路ではなく、最速の経路を使用したいと考えます。通常の道路よりも高速道路の使用を好みます。舗装されていない道よりも舗装されている道路を好みます。実際には、最速の経路と、最短の経路が同じであることはほとんどありません。

## 3.14. プロジェクトジョブのスケジュール

プロジェクトの遅延を最小限に抑えるために、すべてのジョブを時間内に実行できるようにスケジュールを設定します。各ジョブは、プロジェクトに含まれます。ジョブは、異なる方法で実行できます。方法ごとに期間や使用するリソースが異なります。これは、柔軟な **ジョブショップスケジューリング (JSP)** の応用です。

# Project job scheduling

For each job, choose an execution mode and a start time.



ハード制約:

- ジョブの優先順位: ジョブは、先行のジョブがすべて完了するまで開始しない。
- リソースの容量: 利用可能な量を超えるリソースを使用しない。
  - リソースはローカル (同じプロジェクトのジョブ間で共有)、またはグローバル (全ジョブ間で共有) とする。
  - リソースは更新可能 (1 日に利用可能な容量) または更新不可 (全日で利用可能な容量) とする。

中程度の制約:

- プロジェクトの合計遅延時間: 各プロジェクトの所要時間 (メイクスパン) を最短にする。

ソフト制約:

- メイクスパン合計: 複数のプロジェクトスケジュールの合計所要時間を最短にする。

この問題は、[the MISTA 2013 challenge](#) で定義されています。

## 問題の規模

Schedule A-1 has 2 projects, 24 jobs, 64 execution modes, 7 resources and 150 resource requirements.

Schedule A-2 has 2 projects, 44 jobs, 124 execution modes, 7 resources and 420 resource requirements.

Schedule A-3 has 2 projects, 64 jobs, 184 execution modes, 7 resources and 630 resource requirements.

Schedule A-4 has 5 projects, 60 jobs, 160 execution modes, 16 resources and 390 resource requirements.

Schedule A-5 has 5 projects, 110 jobs, 310 execution modes, 16 resources and 900 resource requirements.

Schedule A-6 has 5 projects, 160 jobs, 460 execution modes, 16 resources and 1440 resource requirements.

Schedule A-7 has 10 projects, 120 jobs, 320 execution modes, 22 resources and 900 resource requirements.

Schedule A-8 has 10 projects, 220 jobs, 620 execution modes, 22 resources and 1860 resource requirements.

Schedule A-9 has 10 projects, 320 jobs, 920 execution modes, 31 resources and 2880 resource requirements.

Schedule A-10 has 10 projects, 320 jobs, 920 execution modes, 31 resources and 2970 resource requirements.

Schedule B-1 has 10 projects, 120 jobs, 320 execution modes, 31 resources and 900 resource requirements.

Schedule B-2 has 10 projects, 220 jobs, 620 execution modes, 22 resources and 1740 resource requirements.

Schedule B-3 has 10 projects, 320 jobs, 920 execution modes, 31 resources and 3060 resource requirements.

Schedule B-4 has 15 projects, 180 jobs, 480 execution modes, 46 resources and 1530 resource requirements.

Schedule B-5 has 15 projects, 330 jobs, 930 execution modes, 46 resources and 2760 resource requirements.

Schedule B-6 has 15 projects, 480 jobs, 1380 execution modes, 46 resources and 4500 resource requirements.

Schedule B-7 has 20 projects, 240 jobs, 640 execution modes, 61 resources and 1710 resource requirements.

Schedule B-8 has 20 projects, 440 jobs, 1240 execution modes, 42 resources and 3180 resource requirements.

Schedule B-9 has 20 projects, 640 jobs, 1840 execution modes, 61 resources and 5940 resource requirements.

Schedule B-10 has 20 projects, 460 jobs, 1300 execution modes, 42 resources and 4260 resource requirements.

### 3.15. タスクの割り当て

従業員のキューのスポットに各タスクを割り当てます。タスクごとに、従業員のアフィニティーレベルから影響を受ける期間と、タスクの顧客が含まれます。

ハード制約:

- スキル: タスクごとに1つ以上のスキルが必要である。従業員には、このようなスキルがすべて必要です。

ソフトレベル 0 の制約:

- 極めて重要なタスク: 主要なタスクやマイナーなタスクの前に、極めて重要なタスクを完了する。

ソフトレベル 1 の制約:

- メークスパンの最小化: 全タスクを完了するまでの時間を短縮する。

〜 顧客の長い従業員から順番に始めていき、公平性を保つ。いばニ、バ、バ、ダを、作成する



- 勤務歴の長い従業員から順番に延めし、公平性やロードハフノックを作成する。

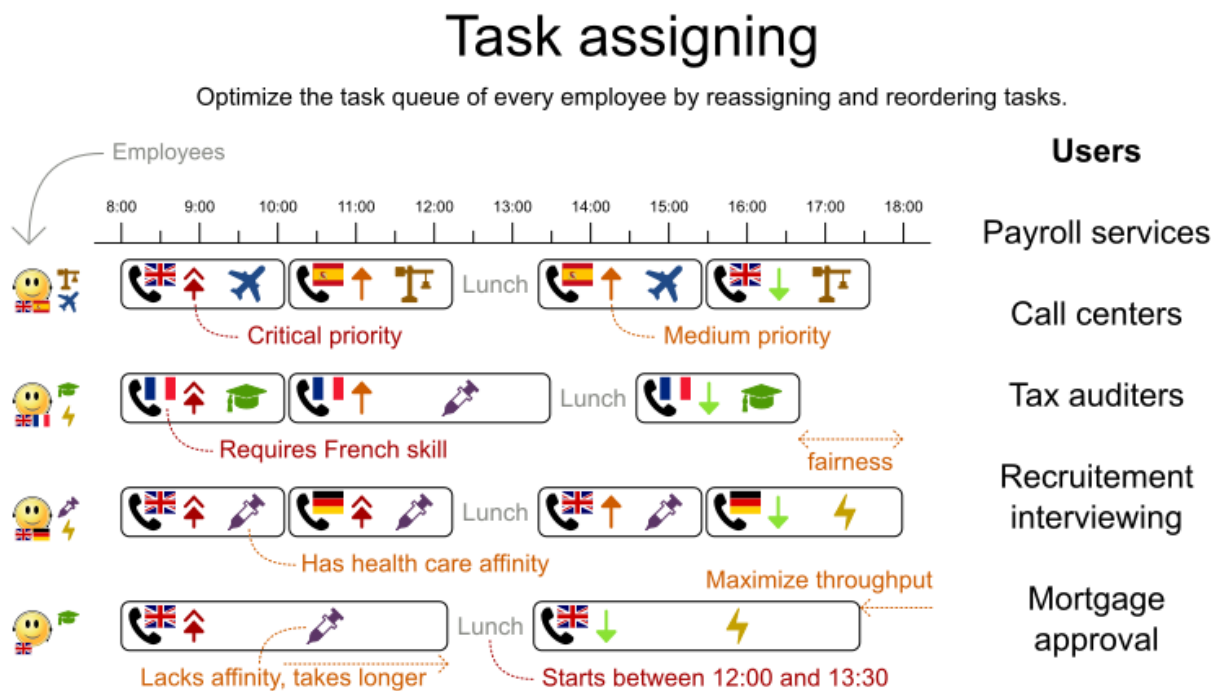
ソフトレベル 2 の制約:

- 主要なタスク: マイナーなタスクの前に、主要なタスクをできるだけ早く完了する。

ソフトレベル 3 の制約:

- マイナーなタスク: できるだけ早くマイナーなタスクを完了する。

図3.9 価値提案



## 問題の規模

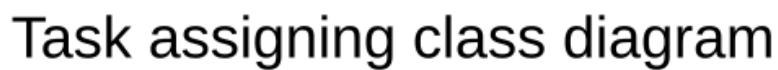
24tasks-8employees has 24 tasks, 6 skills, 8 employees, 4 task types and 4 customers with a search space of  $10^{30}$ .

50tasks-5employees has 50 tasks, 5 skills, 5 employees, 10 task types and 10 customers with a search space of  $10^{69}$ .

100tasks-5employees has 100 tasks, 5 skills, 5 employees, 20 task types and 15 customers with a search space of  $10^{164}$ .

500tasks-20employees has 500 tasks, 6 skills, 20 employees, 100 task types and 60 customers with a search space of  $10^{1168}$ .

### 図3.10 ドメインモデル



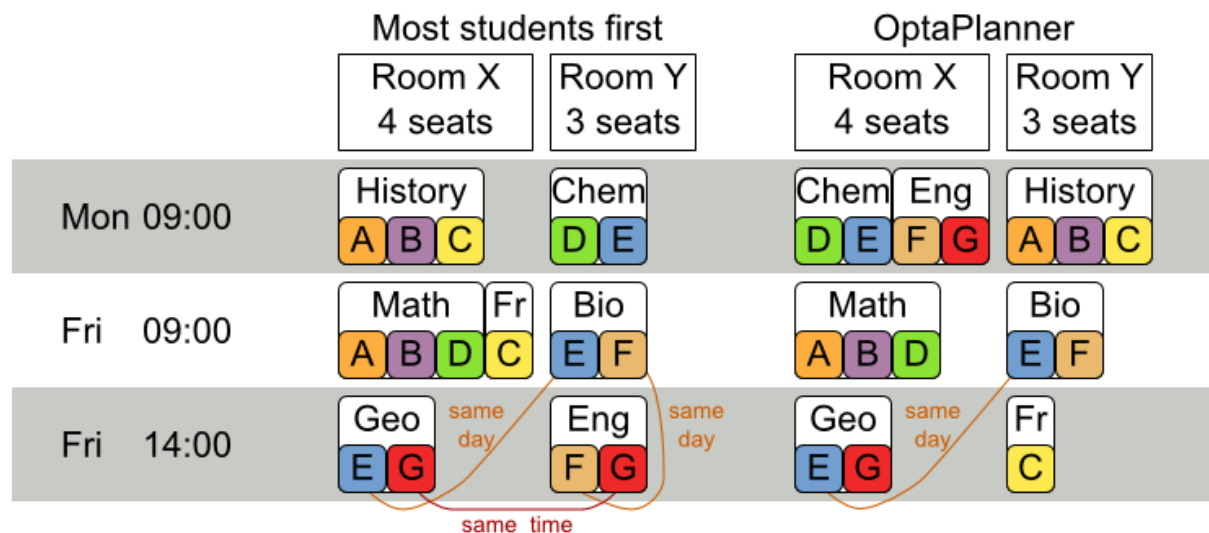
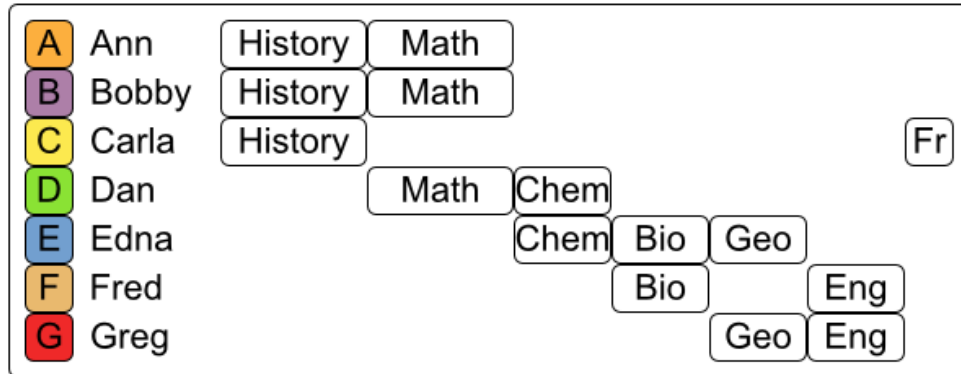
### 3.16. 試験の時間割 (ITC 2007 TRACK 1 - 試験)

すべての試験に、時間と部屋を割り当てます。同じ時間帯に同じ部屋で、複数の試験を行うことができるものとします。



## Examination timetabling

Assign each exam a period and a room.



ハード制約:

- 試験の制約: 同じ学生が受ける 2 つの試験は、同じ時間帯に実施できないものとする。
- 教室の収容人数: 教室の座席数は、常に受験者数よりも多くなければならない。
- 期間: 期間は、すべての試験に対応できる長さでなければならない。
- 期間関連のハード制約 (データセットごとに指定):
  - 一致: 特定の 2 つの試験を同じ時間帯に設定する必要がある (別の教室を使用することも可能)。
  - 除外: 特定の 2 つの試験を同じ時間帯に設定できない。
  - 以降: 特定の試験を、別の特定の試験の後に行う必要がある。
- 教室関連の制約 (データセットごとに指定):
  - 排他的: 特定の試験を、他の試験と同じ教室で行うことはできない。

ソフト制約 (パラメーター化されたペナルティーがそれぞれ設定されている):

- 同じ学生が、続けて試験を 2 つ受けてはいけない。
- 同じ学生が、同じ日に試験を 2 つ受けてはいけない。
- 時間帯の分散: 同じ学生が受ける 2 つの試験は、時間をある程度あける。

- 異なる試験の長さ: 教室を共有する 2 つの試験の長さは、同じにする。
- 前倒し: 規模の大きい試験は、スケジュールを早めに決定する。
- 期間のペナルティー (データセットごとに指定): 期間によっては、使用されるとペナルティーが発生する。
- 部屋のペナルティー (データセットごとに指定): 部屋によっては、使用されるとペナルティーが発生する。

実際に大学から取得した大規模な試験データセットを使用します。

この問題は、[International Timetabling Competition 2007 track 1](#) で定義されています。Geoffrey De Smet は、非常に初期バージョンの OptaPlanner で 4 位を終了しました。このコンペティション以降、多くの改良点が加えられています。

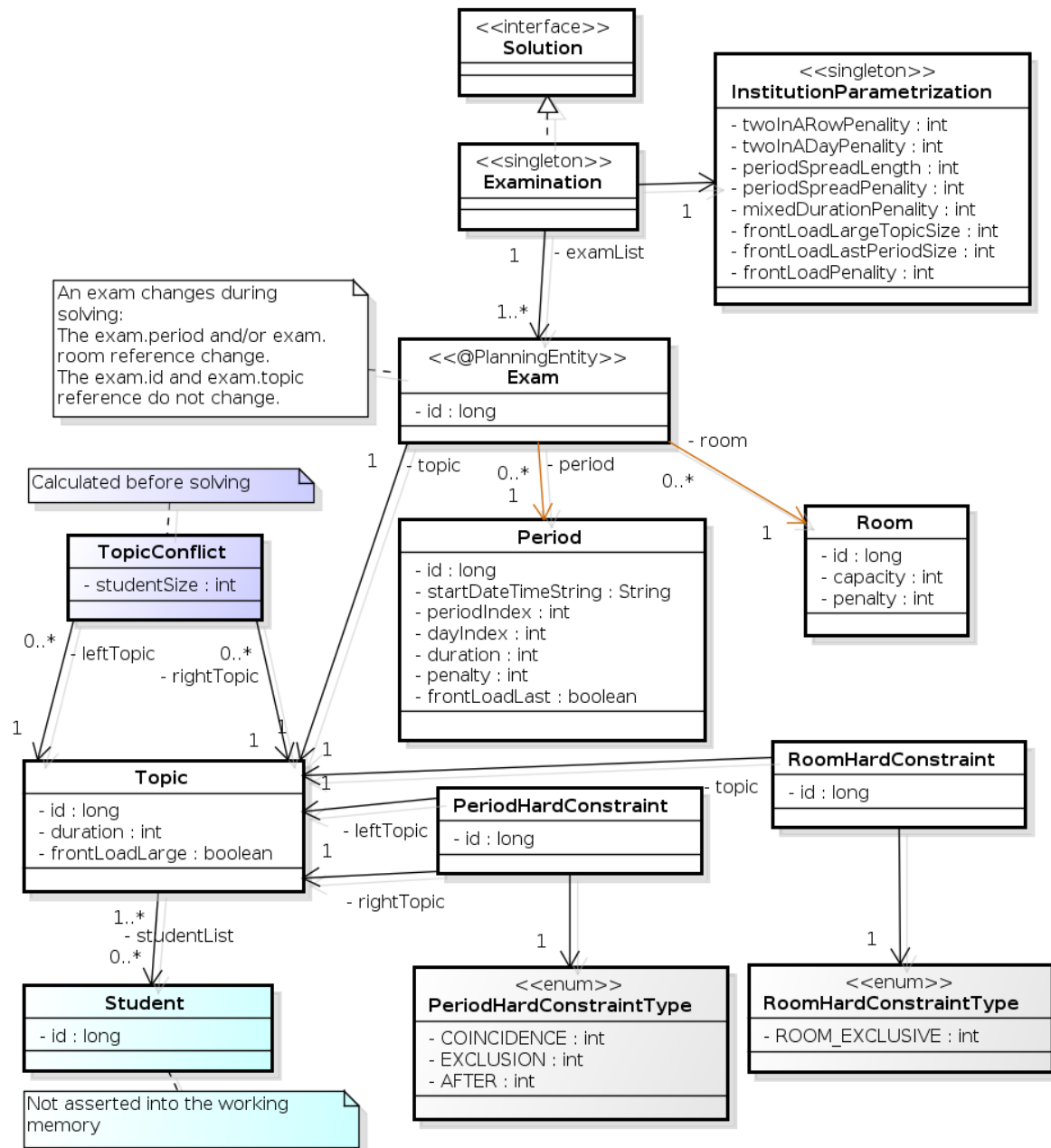
## 問題の規模

```
exam_comp_set1 has 7883 students, 607 exams, 54 periods, 7 rooms, 12 period constraints and
0 room constraints with a search space of 10^1564.
exam_comp_set2 has 12484 students, 870 exams, 40 periods, 49 rooms, 12 period constraints and
2 room constraints with a search space of 10^2864.
exam_comp_set3 has 16365 students, 934 exams, 36 periods, 48 rooms, 168 period constraints and
15 room constraints with a search space of 10^3023.
exam_comp_set4 has 4421 students, 273 exams, 21 periods, 1 rooms, 40 period constraints and
0 room constraints with a search space of 10^360.
exam_comp_set5 has 8719 students, 1018 exams, 42 periods, 3 rooms, 27 period constraints and
0 room constraints with a search space of 10^2138.
exam_comp_set6 has 7909 students, 242 exams, 16 periods, 8 rooms, 22 period constraints and
0 room constraints with a search space of 10^509.
exam_comp_set7 has 13795 students, 1096 exams, 80 periods, 15 rooms, 28 period constraints and
0 room constraints with a search space of 10^3374.
exam_comp_set8 has 7718 students, 598 exams, 80 periods, 8 rooms, 20 period constraints and
1 room constraints with a search space of 10^1678.
```

### 3.16.1. テストの時間割のドメインモデル

以下の図では、主な試験のドメインクラスを紹介しています。

図3.11 試験のドメインクラスの図



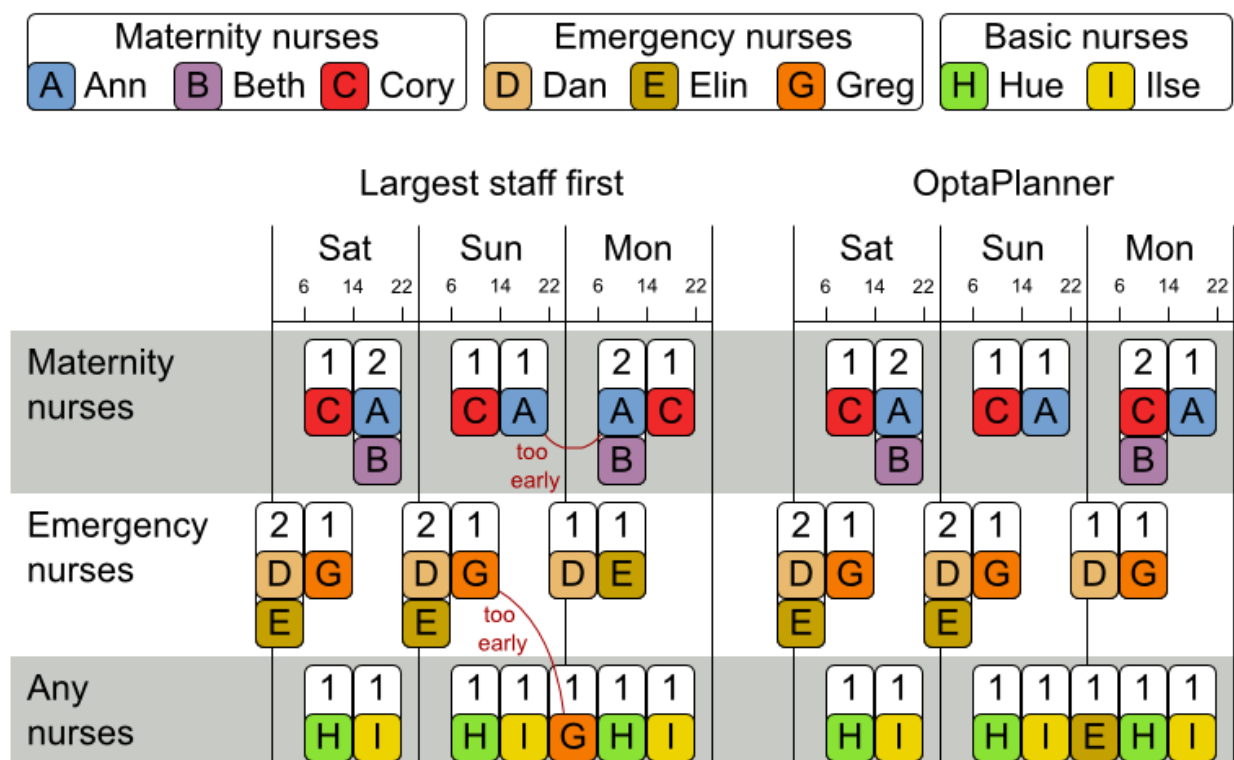
試験のコンセプトを、**Exam** クラスと **Topic** クラスに分けた点に注意してください。期間または教室のプロパティを変更し、解 (プランニングエンティティークラス) を求めると、**Exam** インスタンスが変化します。このとき、**Topic** インスタンス、**Period** インスタンス、および **Room** インスタンスは変化しません (他のクラスと同様、これらも問題ファクトです)。

### 3.17. 看護師の勤務表 (INRC 2010)

各シフトに看護師を割り当てます。

# Employee shift rostering

Populate each work shift with a nurse.



ハード制約:

- **未割り当てのシフトなし (組み込み):** すべてのシフトを従業員に割り当てる必要がある。
- **シフトの制約:** 従業員には1日に1シフトだけ割り当てることができる。

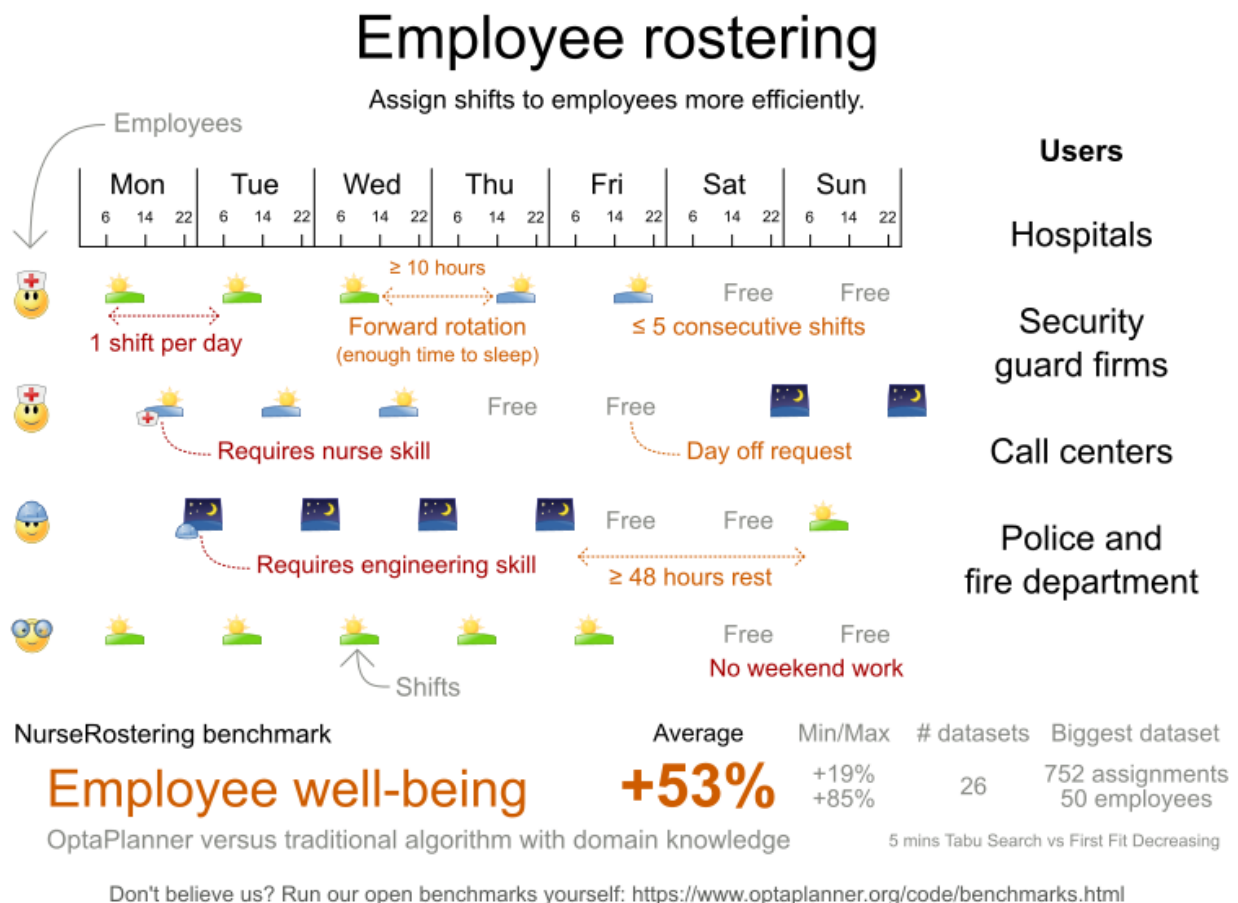
ソフト制約:

- **契約上の義務。** この業界では、頻繁に契約上の義務に違反するため、ハード制約ではなく、ソフト制約として定義することに決定しました。
  - **割り当ての下限および上限:** 各従業員は、(それぞれの契約に合わせて)  $x$  より多く、 $y$  よりも少ないシフト数を勤務する必要がある。
  - **連続勤務日数の下限および上限:** 各従業員は、(それぞれの契約に合わせて) 連続で  $x$  日から  $y$  日間、勤務する必要がある。
  - **連続公休日数の下限および上限:** 各従業員は、(それぞれの契約に合わせて) 連続で  $x$  日から  $y$  日間、休む必要がある。
  - **週末に連続勤務する回数**の下限および上限: 各従業員は、(それぞれの契約に合わせて) 連続で  $x$  回から  $y$  回、週末勤務する必要がある。
  - **週末の勤務有無を同じにする:** 各従業員は、週末の両日を勤務する、または休む必要がある。
  - **週末のシフトタイプを同じにする:** 各従業員で、同じ週末のシフトタイプは、同じにする必要がある。

- 好ましくないシフトパターン: 遅番+早番+遅番など、好ましくないシフトタイプを連続で組み合わせたパターン。
- 従業員の希望:
  - 勤務日のリクエスト: 従業員は、特定の勤務希望日を申請できる。
  - 公休日のリクエスト: 従業員は、特定の公休希望日を申請できる。
  - 勤務するシフトのリクエスト: 従業員は特定のシフトへの割り当てを希望できる。
  - 勤務しないシフトのリクエスト: 従業員は特定のシフトに割り当てられないように希望できる。
- 他のスキル: スキルに割り当てられた従業員は、そのシフトに必要な全スキルに堪能である必要がある。

この問題は [International Nurse Rostering Competition 2010](#) で定義されています。

図3.12 価値提案



## 問題の規模

以下のように、データセットの種類は3つあります。

- sprint: 数秒で問題を解決する必要があります。
- medium: 数分で問題を解決する必要があります。
- long: 時間で解決する必要があります。

toy1 has 1 skills, 3 shiftTypes, 2 patterns, 1 contracts, 6 employees, 7 shiftDates, 35 shiftAssignments and 0 requests with a search space of  $10^{27}$ .  
 toy2 has 1 skills, 3 shiftTypes, 3 patterns, 2 contracts, 20 employees, 28 shiftDates, 180 shiftAssignments and 140 requests with a search space of  $10^{234}$ .

sprint01 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint02 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint03 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint04 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint05 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint06 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint07 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint08 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint09 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint10 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint\_hint01 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint\_hint02 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint\_hint03 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint\_late01 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint\_late02 has 1 skills, 3 shiftTypes, 4 patterns, 3 contracts, 10 employees, 28 shiftDates, 144 shiftAssignments and 139 requests with a search space of  $10^{144}$ .  
 sprint\_late03 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 160 shiftAssignments and 150 requests with a search space of  $10^{160}$ .  
 sprint\_late04 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 160 shiftAssignments and 150 requests with a search space of  $10^{160}$ .  
 sprint\_late05 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint\_late06 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint\_late07 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .  
 sprint\_late08 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 0 requests with a search space of  $10^{152}$ .  
 sprint\_late09 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 0 requests with a search space of  $10^{152}$ .  
 sprint\_late10 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of  $10^{152}$ .

medium01 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of  $10^{906}$ .  
 medium02 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of  $10^{906}$ .

medium03 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of  $10^9$ 06.

medium04 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of  $10^9$ 06.

medium05 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of  $10^9$ 06.

medium\_hint01 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of  $10^6$ 32.

medium\_hint02 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of  $10^6$ 32.

medium\_hint03 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of  $10^6$ 32.

medium\_late01 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 424 shiftAssignments and 390 requests with a search space of  $10^6$ 26.

medium\_late02 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of  $10^6$ 32.

medium\_late03 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of  $10^6$ 32.

medium\_late04 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 416 shiftAssignments and 390 requests with a search space of  $10^6$ 14.

medium\_late05 has 2 skills, 5 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 452 shiftAssignments and 390 requests with a search space of  $10^6$ 67.

long01 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of  $10^{12}$ 50.

long02 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of  $10^{12}$ 50.

long03 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of  $10^{12}$ 50.

long04 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of  $10^{12}$ 50.

long05 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of  $10^{12}$ 50.

long\_hint01 has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and 0 requests with a search space of  $10^{12}$ 57.

long\_hint02 has 2 skills, 5 shiftTypes, 7 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and 0 requests with a search space of  $10^{12}$ 57.

long\_hint03 has 2 skills, 5 shiftTypes, 7 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and 0 requests with a search space of  $10^{12}$ 57.

long\_late01 has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and 0 requests with a search space of  $10^{12}$ 77.

long\_late02 has 2 skills, 5 shiftTypes, 9 patterns, 4 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and 0 requests with a search space of  $10^{12}$ 77.

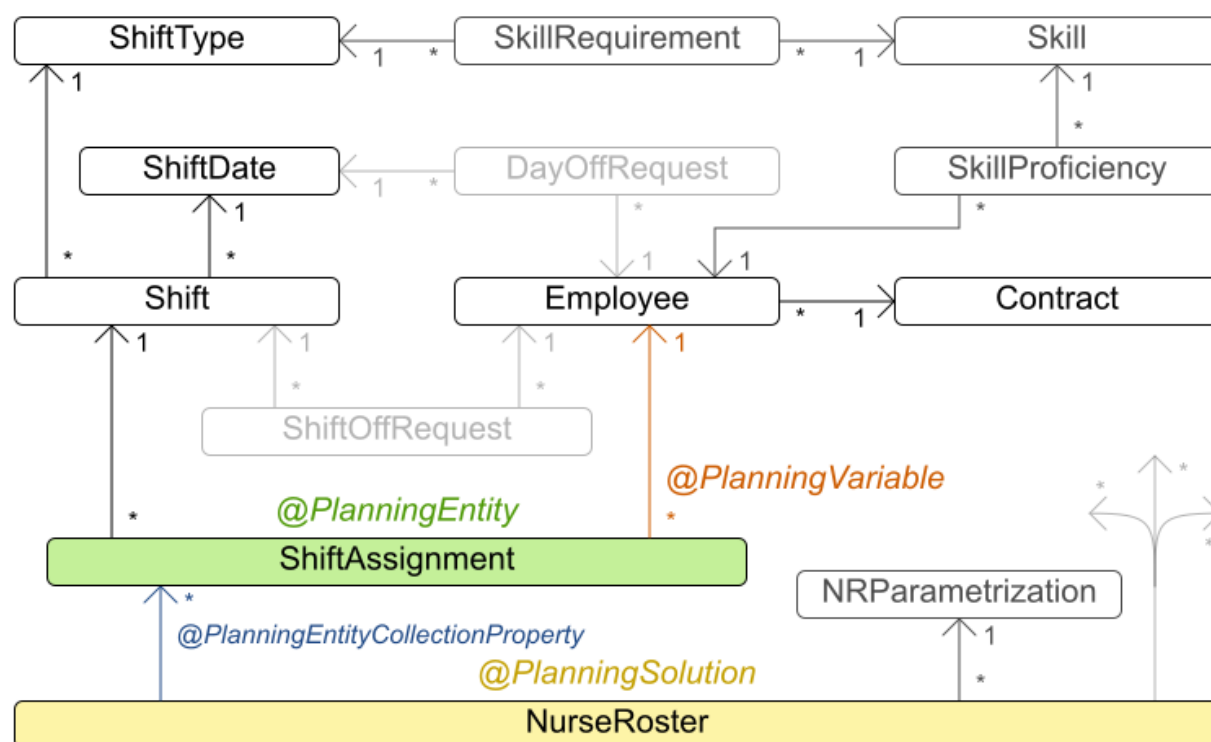
long\_late03 has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and 0 requests with a search space of  $10^{12}$ 77.

long\_late04 has 2 skills, 5 shiftTypes, 9 patterns, 4 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and 0 requests with a search space of  $10^{12}$ 77.

long\_late05 has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and 0 requests with a search space of  $10^{12}$ 57.

図3.13 ドメインモデル

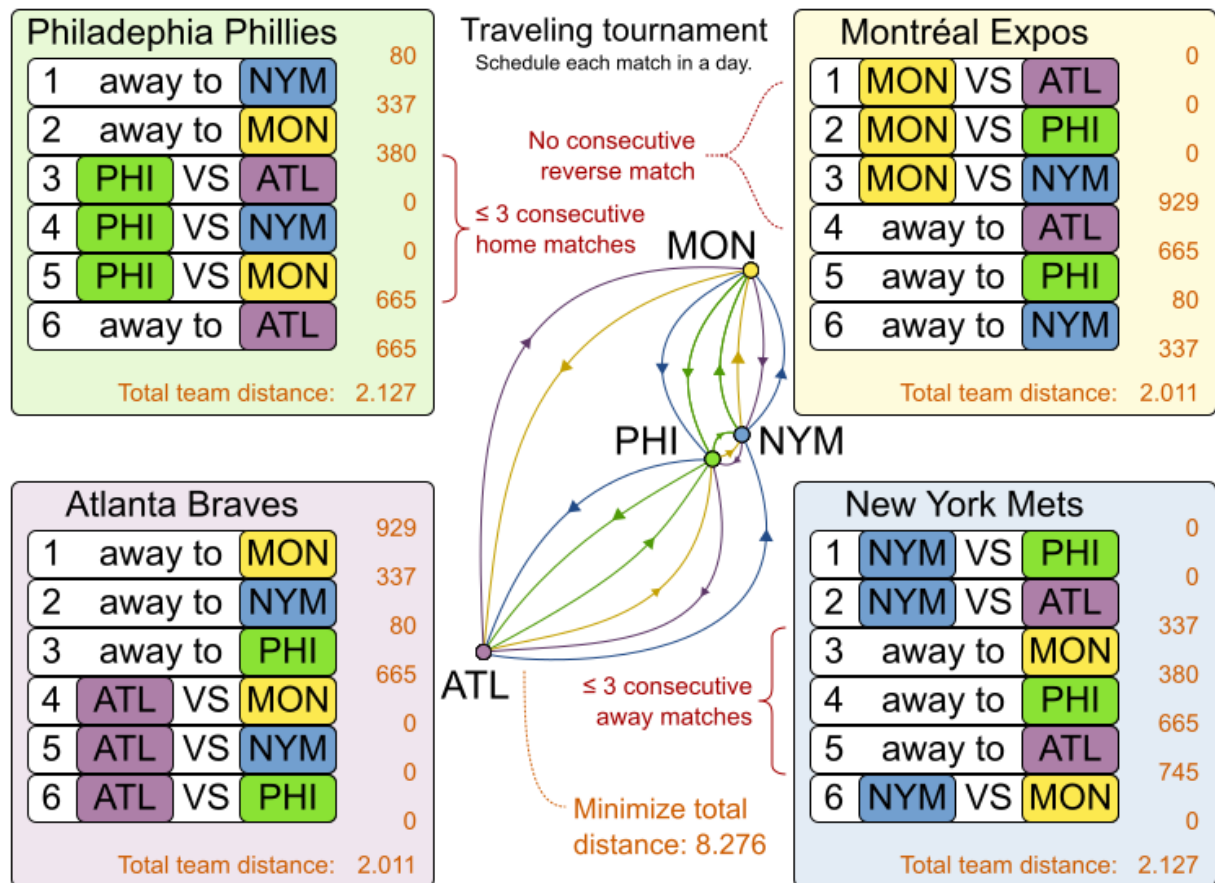
## Nurse rostering class diagram



## 3.18. 巡回トーナメント問題 (TTP)

n 人数のチーム間の一致をスケジュールします。





ハード制約:

- 各チームは、他のチームとそれぞれ2回(ホームとアウェイ)試合をする。
- 各チームは、各時間枠に1試合だけ行う。
- 3回連続で、ホームまたはアウェイでの試合はできない。
- 繰り返しなし: 同じ対戦相手と2回連続で対戦できない。

ソフト制約:

- 全チームが移動する合計距離を最小限に抑える。

この問題は [Michael Trick の Web サイト \(世界記録が含まれます\)](#) で定義されています。

## 問題の規模

1-nl04	has 6 days, 4 teams and 12 matches with a search space of	$10^5$ .
1-nl06	has 10 days, 6 teams and 30 matches with a search space of	$10^{19}$ .
1-nl08	has 14 days, 8 teams and 56 matches with a search space of	$10^{43}$ .
1-nl10	has 18 days, 10 teams and 90 matches with a search space of	$10^{79}$ .
1-nl12	has 22 days, 12 teams and 132 matches with a search space of	$10^{126}$ .
1-nl14	has 26 days, 14 teams and 182 matches with a search space of	$10^{186}$ .
1-nl16	has 30 days, 16 teams and 240 matches with a search space of	$10^{259}$ .
2-bra24	has 46 days, 24 teams and 552 matches with a search space of	$10^{692}$ .
3-nfl16	has 30 days, 16 teams and 240 matches with a search space of	$10^{259}$ .
3-nfl18	has 34 days, 18 teams and 306 matches with a search space of	$10^{346}$ .

```

3-nfl20  has 38 days, 20 teams and 380 matches with a search space of 10^447.
3-nfl22  has 42 days, 22 teams and 462 matches with a search space of 10^562.
3-nfl24  has 46 days, 24 teams and 552 matches with a search space of 10^692.
3-nfl26  has 50 days, 26 teams and 650 matches with a search space of 10^838.
3-nfl28  has 54 days, 28 teams and 756 matches with a search space of 10^999.
3-nfl30  has 58 days, 30 teams and 870 matches with a search space of 10^1175.
3-nfl32  has 62 days, 32 teams and 992 matches with a search space of 10^1367.
4-super04 has 6 days, 4 teams and 12 matches with a search space of 10^5.
4-super06 has 10 days, 6 teams and 30 matches with a search space of 10^19.
4-super08 has 14 days, 8 teams and 56 matches with a search space of 10^43.
4-super10 has 18 days, 10 teams and 90 matches with a search space of 10^79.
4-super12 has 22 days, 12 teams and 132 matches with a search space of 10^126.
4-super14 has 26 days, 14 teams and 182 matches with a search space of 10^186.
5-galaxy04 has 6 days, 4 teams and 12 matches with a search space of 10^5.
5-galaxy06 has 10 days, 6 teams and 30 matches with a search space of 10^19.
5-galaxy08 has 14 days, 8 teams and 56 matches with a search space of 10^43.
5-galaxy10 has 18 days, 10 teams and 90 matches with a search space of 10^79.
5-galaxy12 has 22 days, 12 teams and 132 matches with a search space of 10^126.
5-galaxy14 has 26 days, 14 teams and 182 matches with a search space of 10^186.
5-galaxy16 has 30 days, 16 teams and 240 matches with a search space of 10^259.
5-galaxy18 has 34 days, 18 teams and 306 matches with a search space of 10^346.
5-galaxy20 has 38 days, 20 teams and 380 matches with a search space of 10^447.
5-galaxy22 has 42 days, 22 teams and 462 matches with a search space of 10^562.
5-galaxy24 has 46 days, 24 teams and 552 matches with a search space of 10^692.
5-galaxy26 has 50 days, 26 teams and 650 matches with a search space of 10^838.
5-galaxy28 has 54 days, 28 teams and 756 matches with a search space of 10^999.
5-galaxy30 has 58 days, 30 teams and 870 matches with a search space of 10^1175.
5-galaxy32 has 62 days, 32 teams and 992 matches with a search space of 10^1367.
5-galaxy34 has 66 days, 34 teams and 1122 matches with a search space of 10^1576.
5-galaxy36 has 70 days, 36 teams and 1260 matches with a search space of 10^1801.
5-galaxy38 has 74 days, 38 teams and 1406 matches with a search space of 10^2042.
5-galaxy40 has 78 days, 40 teams and 1560 matches with a search space of 10^2301.

```

### 3.19. コストを抑えるスケジュール

全タスクを時間内にスケジュールし、機械の電気代を最小限に抑えます。電気代は時間によって異なります。これは、**ジョブショップスケジューリング**の応用です。

ハード制約:

- 開始時間の制限: 各タスクは、最早と最遅の開始時間の制限内に、開始する必要があります。
- 最大容量: マシンに割り当てる各リソースはこの量を超えてはいけません。
- 開始および終了: 各機械は、タスクが割り当てられている間は稼働している必要があります。次のタスクまでの間、起動および終了コストを避けるため、機械をアイドルにすることができます。

中程度の制約:

- 電気代: 全スケジュールの合計電気代を最小限に抑える。
  - 機械の電気代: 稼働中またはアイドル中の機械はそれぞれ、電気を消費し、電気代が発生する (金額は使用時の電気代によって異なる)。
  - タスクの電気代: 各タスクも電気を消費し、電気代が発生する (金額は使用時の電気代によって異なる)。

- 機械の起動および終了コスト: 機械を起動または終了するたびに、追加のコストが発生する。

ソフト制約 (問題に元々設定されている定義に追加):

- 早く開始: なるべく早めにタスクを開始するようにする。

この問題は、[ICON challenge](#) で定義されています。

## 問題の規模

```
sample01 has 3 resources, 2 machines, 288 periods and 25 tasks with a search space of 10^53.
sample02 has 3 resources, 2 machines, 288 periods and 50 tasks with a search space of 10^114.
sample03 has 3 resources, 2 machines, 288 periods and 100 tasks with a search space of 10^226.
sample04 has 3 resources, 5 machines, 288 periods and 100 tasks with a search space of 10^266.
sample05 has 3 resources, 2 machines, 288 periods and 250 tasks with a search space of 10^584.
sample06 has 3 resources, 5 machines, 288 periods and 250 tasks with a search space of 10^673.
sample07 has 3 resources, 2 machines, 288 periods and 1000 tasks with a search space of 10^2388.
sample08 has 3 resources, 5 machines, 288 periods and 1000 tasks with a search space of 10^2748.
sample09 has 4 resources, 20 machines, 288 periods and 2000 tasks with a search space of 10^6668.
instance00 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^595.
instance01 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^599.
instance02 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^599.
instance03 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^591.
instance04 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^590.
instance05 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^667.
instance06 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^660.
instance07 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^662.
instance08 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^651.
instance09 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^659.
instance10 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1657.
instance11 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1644.
instance12 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1637.
instance13 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of
```

10<sup>1659</sup>.  
instance14 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10<sup>1643</sup>.  
instance15 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10<sup>1782</sup>.  
instance16 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10<sup>1778</sup>.  
instance17 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10<sup>1764</sup>.  
instance18 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10<sup>1769</sup>.  
instance19 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10<sup>1778</sup>.  
instance20 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10<sup>3689</sup>.  
instance21 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10<sup>3678</sup>.  
instance22 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10<sup>3706</sup>.  
instance23 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10<sup>3676</sup>.  
instance24 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10<sup>3681</sup>.  
instance25 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10<sup>3774</sup>.  
instance26 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10<sup>3737</sup>.  
instance27 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10<sup>3744</sup>.  
instance28 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10<sup>3731</sup>.  
instance29 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10<sup>3746</sup>.  
instance30 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10<sup>7718</sup>.  
instance31 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10<sup>7740</sup>.  
instance32 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10<sup>7686</sup>.  
instance33 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10<sup>7672</sup>.  
instance34 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10<sup>7695</sup>.  
instance35 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10<sup>7807</sup>.  
instance36 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10<sup>7814</sup>.  
instance37 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10<sup>7764</sup>.  
instance38 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10<sup>7736</sup>.  
instance39 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10<sup>7783</sup>.  
instance40 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of 10<sup>15976</sup>.  
instance41 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of

```

10^15935.
instance42 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of
10^15887.
instance43 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of
10^15896.
instance44 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of
10^15885.
instance45 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20173.
instance46 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20132.
instance47 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20126.
instance48 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20110.
instance49 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20078.

```

### 3.20. 投資資産クラスの割り当て (ポートフォリオの最適化)

各資産クラスに投資する相対数を決定します。

ハード制約:

- リスクの最大値: 標準偏差合計は、標準偏差の最大値を超えてはならない。
  - 標準偏差合計の計算は、[Markowitz Portfolio Theory](#) を適用した、資産クラスの相対関係を考慮する必要がある。
- 地域の最大値: 地域ごとに数量の最大値がある。
- セクターの最大値: 各セクターに数量の最大値がある。

ソフト制約:

- 期待収益を最大化する。

#### 問題の規模

```

de_smet_1 has 1 regions, 3 sectors and 11 asset classes with a search space of 10^4.
irrinki_1 has 2 regions, 3 sectors and 6 asset classes with a search space of 10^3.

```

サイズが大きいデータセットは作成/検証されていませんが、問題はないはずです。データに関する適切な情報源として、[このアセット関連の Web サイト](#) を参照してください。

### 3.21. 会議スケジュール

各会議を時間帯と部屋に割り当てていきます。時間帯は重複させることができます。LibreOffice や Excel で編集可能な \*.xlsx ファイルとの読み書きが可能です。

ハード制約:

- 時間帯の会議タイプ: 会議のタイプは、時間帯の会議タイプと一致する必要がある。
- 部屋が使用中の時間帯: その会議の時間帯に、会議用の部屋が利用できない。

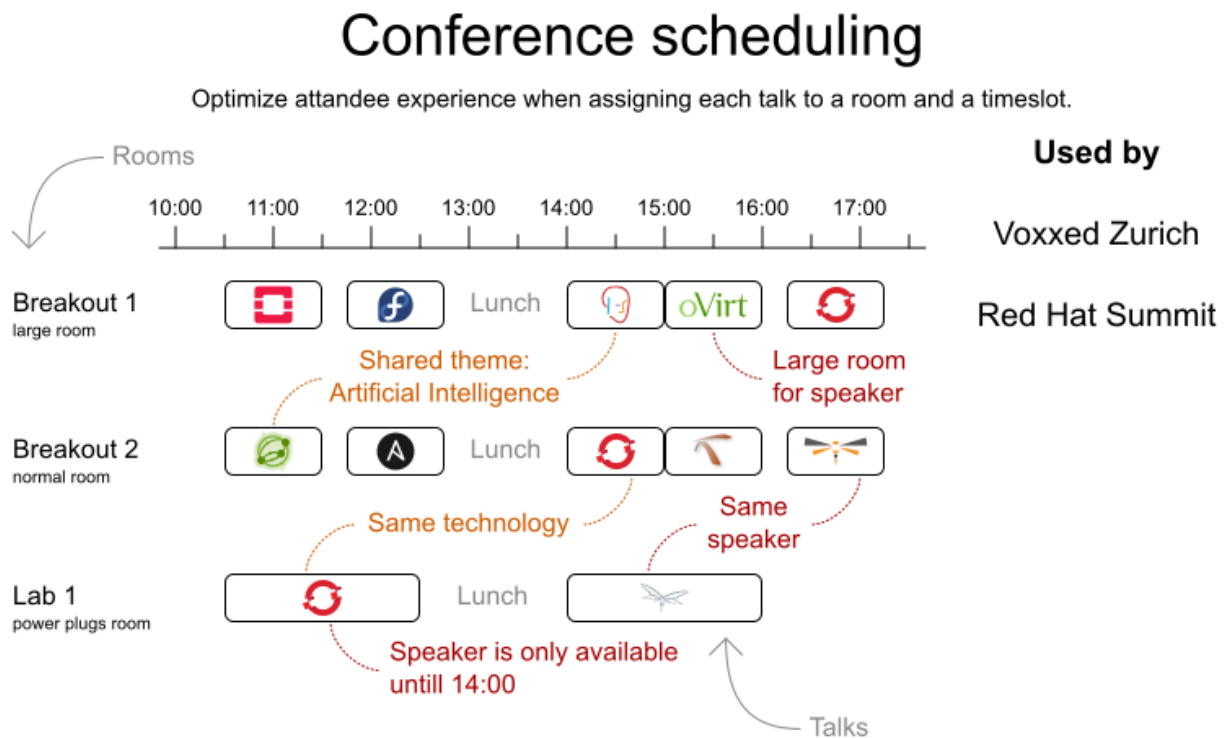
- 部屋の競合: 2つの会議が、同じ時間に同じ会議室を使用することはできない。
- 講演者が空いていない時間帯: 講演者は必ず、会議の時間帯に空いていなければならない。
- 講演者の競合: 同じ時間帯の2つの会議に同じ講演者を割り当てることができない。
- 汎用の時間帯および部屋タグ:
  - 講演者が要求する時間帯タグ: 講演者に、必須時間帯タグが付けられている場合、講演者の会議はそのタグが付いている時間に割り当てる必要がある。
  - 講演者の禁止時間帯タグ: 公演者に、禁止時間帯タグが割り当てられている場合は、そのタグの付いた時間帯に講演者の会議をどれも割り当てることができない。
  - 会議を設定する必要がある時間帯タグ: 会議に必須時間帯タグが付いている場合は、そのタグの付いた時間帯に割り当てる必要がある。
  - 会議の禁止時間帯タグ: 会議に、禁止時間帯タグが割り当てられている場合は、そのタグの付いた時間帯にその会議を割り当てることができない。
  - 講演者が要求する部屋のタグ: 講演者に、必須の部屋タグが付けられている場合、講演者の会議はそのタグが付いている部屋に割り当てる必要がある。
  - 講演者が禁止する部屋のタグ: 講演者に、禁止部屋のタグが付けられている場合、講演者の会議はそのタグが付いている部屋に割り当てることができない。
  - 会議を設定する必要がある部屋タグ: 会議に必須部屋タグが付いている場合は、そのタグの付いた部屋に割り当てる必要がある。
  - 会議の禁止部屋タグ: 会議に、禁止部屋タグが割り当てられている場合は、そのタグの付いた部屋にその会議を割り当てることができない。
- 他の会議と同じ時間帯に設定しないタグ: このタグが付いている会議は、同じ時間帯に重複してスケジュールしてはいけない。
- 受講条件が付いた会議: 受講条件が付いた会議をすべて完了してからでないと対象の会議をスケジュールしてはいけない。

#### ソフト制約:

- テーマの追跡競合: 同じ時間帯で、テーマのタグが付いた会議の数を最小限に抑える。
- セクターの競合: 同じ時間帯で同じセクタータグの付いた会議の数を最小限に抑える。
- コンテンツの受講者レベルのフロー違反: すべてのコンテンツタグに対して、上級者用の会議の前に入門レベルの会議をスケジュールする。
- 受講者レベルの多様性: すべての時間帯において、異なる受講者レベルの会議数を最大限に増やす。
- 言語の多様性: すべての時間帯において、異なる言語の会議数を最大限を増やす。
- 汎用の時間帯および部屋タグ:
  - 講演者が希望する時間帯タグ: 講演者に、希望の時間帯タグが付けられている場合、講演者の会議はそのタグが付いている時間に割り当てるようにする。

- 講演者が希望しないタイムスロットタグ: 講演者が望ましくないタイムスロットタグを持っている場合、そのタグが付いているタイムスロットには講演を割り当ててはいけません。
- 会議の希望の時間帯タグ: 会議に希望の時間帯タグが付いている場合は、そのタグの付いた時間帯に割り当てるようにする。
- 会議の設定を希望しない時間帯タグ: 会議に、希望しない時間帯タグが付いている場合は、そのタグの付いた時間帯に割り当てないようにする。
- 講演者が希望する部屋のタグ: 講演者に、希望の部屋タグが付けられている場合、講演者の会議はそのタグが付いている部屋に割り当てるようにする。
- 講演者が希望しない部屋のタグ: 講演者に、希望しない部屋タグが付けられている場合、講演者の会議はそのタグが付いている部屋に割り当てないようにする。
- 会議を希望の部屋タグ: 会議に希望の部屋タグが付いている場合は、そのタグの付いた部屋に割り当てるようにする。
- 会議での使用を希望しない部屋タグ: 会議に、希望しない部屋タグが付いている場合、そのタグの付いた部屋に割り当てないようにする。
- 同じ日の会議: テーマタグまたはコンテンツタグを共有する会議は、最低限の日数 (理想的には同じ日) にスケジュールする必要がある。

図3.14 価値提案



### 問題の規模

18talks-6timeslots-5rooms has 18 talks, 6 timeslots and 5 rooms with a search space of  $10^{26}$ .

36talks-12timeslots-5rooms has 36 talks, 12 timeslots and 5 rooms with a search space of  $10^{64}$ .  
 72talks-12timeslots-10rooms has 72 talks, 12 timeslots and 10 rooms with a search space of  $10^{149}$ .  
 108talks-18timeslots-10rooms has 108 talks, 18 timeslots and 10 rooms with a search space of  $10^{243}$ .  
 216talks-18timeslots-20rooms has 216 talks, 18 timeslots and 20 rooms with a search space of  $10^{552}$ .

### 3.22. ロックツアー

次のショーへの移動はロックバンクバスを使用し、空いている日のみショーをスケジュールする。

ハード制約:

- 必要とされるショーをすべてスケジュールする。
- できるだけ多くのショーをスケジュールする。

中程度の制約:

- 収益の機会を最大化する。
- 運転時間を最小限に抑える。
- できるだけ早く到着する。

ソフト制約:

- 長時間の運転は避ける。

#### 問題の規模

47shows has 47 shows with a search space of  $10^{59}$ .

### 3.23. 航空機乗組員のスケジューリング

パイロットと客室乗務員にフライトを割り当てます。

ハード制約:

- 必須スキル: フライトの割り当てにはそれぞれ、必要とされるスキルがあります。たとえば、フライト AB0001 ではパイロット 2 名と、客室乗務員 3 名が必要です。
- フライトの競合: 各従業員は同じ時間に出勤できるフライトは 1 つだけにします。
- 2 つのフライト間での移動: 2 つのフライトの間で、従業員は到着先の空港と、出発元の空港に移動できる必要があります。たとえば、アンは 10 時にブリュッセルに到着し、15 時にアムステルダムを出発するなどです。
- 従業員の勤務できない日: 従業員はフライトの当日は空いていなければならない。たとえば、アンは 2 月 1 日に休暇を取っているなど。

ソフト制約:

- 最初の仕事が自宅から出発する。



- 最後の仕事が自宅に到着する。
- 総フライト時間を従業員別に平均的に分散する。

## 問題の規模

175flights-7days-Europe has 2 skills, 50 airports, 150 employees, 175 flights and 875 flight assignments with a search space of  $10^{1904}$ .  
700flights-28days-Europe has 2 skills, 50 airports, 150 employees, 700 flights and 3500 flight assignments with a search space of  $10^{7616}$ .  
875flights-7days-Europe has 2 skills, 50 airports, 750 employees, 875 flights and 4375 flight assignments with a search space of  $10^{12578}$ .  
175flights-7days-US has 2 skills, 48 airports, 150 employees, 175 flights and 875 flight assignments with a search space of  $10^{1904}$ .

## 第4章 RED HAT ビルドの OPTAPLANNER の例のダウンロード

Red Hat ビルドの OptaPlanner のサンプルは、Red Hat カスタマーポータルで利用可能な Red Hat Decision Manager アドオンパッケージの一部としてダウンロードできます。

### 手順

1. Red Hat カスタマーポータルの [Software Downloads](#) ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
  - Product: Decision Manager
  - Version: 7.12
2. Red Hat Decision Manager 7.12 Add Onsをダウンロードします。
3. **rhdm-7.12.0-add-ons.zip** ファイルを展開します。展開した **add-ons** フォルダには、**rhdm-7.12.0-planner-engine.zip** ファイルが含まれます。
4. **rhdm-7.12.0-planner-engine.zip** ファイルを展開します。

### 結果

展開した **rhdm-7.12.0-planner-engine** ディレクトリーには、以下のサブディレクトリー内にサンプルソースコードのサンプルが含まれます。

- **examples/sources/src/main/java/org/optaplanner/examples**
- **examples/sources/src/main/resources/org/optaplanner/examples**

## 4.1. OPTAPLANNER サンプルの実行

Red Hat ビルドの OptaPlanner には、さまざまな計画のユースケースのデモとしてサンプルが複数含まれています。この例をダウンロードして使用し、さまざまなタイプのプランニングソリューションを確認します。

### 前提条件

- [4章 Red Hat ビルドの OptaPlanner の例のダウンロード](#) で記載されているように例をダウンロードして展開している。

### 手順

1. サンプルを実行するには、**rhdm-7.12.0-planner-engine/examples** ディレクトリーで以下のコマンドのいずれかを入力します。

Linux または Mac の場合:

```
$ ./runExamples.sh
```

Windows:

```
$ runExamples.bat
```

OptaPlanner Example ウィンドウが開きます。

2. 実行するサンプルを選択します。



#### 注記

Red Hat ビルドの OptaPlanner は GUI に依存しません。デスクトップと同じように、サーバーまたはモバイル JVM 上でも実行できます。

## 4.2. IDE (INTELLIJ、ECLIPSE、または NETBEANS) での RED HAT ビルドの OPTAPLANNER サンプルの実行

IntelliJ、Eclipse、Netbeans など統合開発環境 (IDE) を使用する場合は、お使いの開発環境にダウンロードした OptaPlanner の例を実行できます。

### 前提条件

- [4章 Red Hat ビルドの OptaPlanner の例のダウンロード](#) の説明に従って、OptaPlanner のサンプルをダウンロードして展開している。

### 手順

1. OptaPlanner の例を新規プロジェクトとして開きます。
  - a. IntelliJ または Netbeans の場合は、新規プロジェクトとして **examples/sources/pom.xml** を開きます。Maven 統合の指示に従い、インストールを進めてください。この手順では、残りのステップは省略します。
  - b. Eclipse の場合は、**rhdm-7.12.0-planner-engine** ディレクトリーにある **/examples/binaries** ディレクトリーの新規プロジェクトを開きます。
2. **binaries** ディレクトリーにあるすべての JAR ファイルをクラスパスに追加します (**examples/binaries/optaplanner-examples-7.59.0.Final-redhat-00006.jar** ファイルを除く)。
3. **rhdm-7.12.0-planner-engine /examples/sources /** ディレクトリーにある Java ソースディレクトリー **src/main/java** ディレクトリーと Java リソースディレクトリー **src/main/resources** を追加します。
4. 実行設定を作成します。
  - メインクラス: **org.optaplanner.examples.app.OptaPlannerExamplesApp**
  - VM パラメーター (任意): **-Xmx512M -server -Dorg.optaplanner.examples.dataDir=examples/sources/data**
  - 作業ディレクトリー: **examples/sources**
5. 実行設定を実行します。

## 第5章 BUSINESS CENTRAL での OPTAPLANNER のスタートガイド: 従業員勤務表の例

Business Central で **employee-rostering** のサンプルプロジェクトを構築して、デプロイできます。このプロジェクトは、シフト勤務の計画問題を解決するのに必要な Business Central の各アセットを作成し、Red Hat ビルドの OptaPlanner を使用して実現可能な最適解を見つける方法を示します。

Business Central に事前設定された **employee-rostering** プロジェクトをデプロイすることができます。Business Central を使用してプロジェクトを独自に作成することができます。



### 注記

Business Central の **employee-rostering** サンプルプロジェクトには、データセットが含まれていません。REST API 呼び出しを使用して XML 形式のデータセットを提供する必要があります。

### 5.1. BUSINESS CENTRAL への従業員勤務表サンプルプロジェクトのデプロイメント

Business Central には、製品と機能に慣れるために使用できるサンプルプロジェクトが多数あります。従業員の勤務表サンプルプロジェクトは、Red Hat ビルドの OptaPlanner でシフト勤務のユースケースを示すために計画され作成されました。以下の手順に従って、従業員の勤務表サンプルを Business Central にデプロイして実行します。

#### 前提条件

- Red Hat Decision Manager をダウンロードしてインストールしている。インストールオプションは [Red Hat Decision Manager インストールの計画](#) を参照してください。
- インストールのドキュメントにあるように、Red Hat Decision Manager が起動し、**admin** パーミッションを持つユーザーとして Business Central にログインしています。

#### 手順

- Business Central で **Menu → Design → Projects** の順にクリックします。
- 事前に設定した **MySpace** スペースで **Try Samples** をクリックします。
- サンプルプロジェクトの一覧から **employee-rostering** を選択し、右上の **OK** をクリックして、プロジェクトをインポートします。
- アセットリストをコンパイルし、**Build & Deploy** をクリックして、従業員の勤務表サンプルをデプロイします。

本書は、各プロジェクトアセットとその設定について説明します。

### 5.2. 従業員の勤務表サンプルプロジェクトの再作成

従業員の勤務表サンプルプロジェクトは、Business Central で使用できる事前設定のプロジェクトです。このプロジェクトをデプロイする方法は、[「Business Central への従業員勤務表サンプルプロジェクトのデプロイメント」](#) を参照してください。

ゼロから従業員勤務表を作成できます。この例では、ワークフローを使用して、Business Central で独自の類似プロジェクトを作成します。

### 5.2.1. 従業員の勤務表プロジェクトの設定

Business Central で Solver の開発を始めるには、プロジェクトを設定する必要があります。

#### 前提条件

- Red Hat Decision Manager をダウンロードしてインストールしている。
- Business Central をデプロイし、**admin** ロールを持つユーザーでログインしている。

#### 手順

1. **Menu → Design → Projects → Add Project** をクリックして、Business Central に新しいプロジェクトを作成します。
2. **Add Project** ウィンドウで、以下のフィールドに入力します。

- **Name:** **employee-rostering**
- **Description** (任意): OptaPlanner を使用した従業員の勤務表問題の最適化。スキルに基づいて、従業員をシフトに割り当てます。

任意で、**Configure Advanced Options** をクリックして、**Group ID**、**Artifact ID**、および **Version** に情報を追加します。

- **Group ID:** **employee-rostering**
  - **Artifact ID:** **employee-rostering**
  - **Version:** **1.0.0-SNAPSHOT**
3. **Add** をクリックして、Business Central プロジェクトリポジトリにプロジェクトを追加します。

### 5.2.2. プロジェクトファクトおよびプランニングエンティティ

従業員勤務表の計画問題の各ドメインクラスは、以下のいずれかに分類されます。

- **関連性のないクラス:** どのスコア制約にも使用されません。計画に関して言えば、このデータは使用されません。
- **問題ファクト クラス:** スコア制約に使用されますが、(問題が変わらない限り) 計画時には変化しません (例: **Shift**、**Employee**)。問題ファクトクラスのプロパティはすべて問題のプロパティです。
- **プランニングエンティティークラス クラス:** スコア制約に使用され、計画時に変化します (例: **ShiftAssignment**)。計画時に変更するプロパティは **プランニング変数** です。その他のプロパティは問題プロパティです。  
以下の点についてお考え下さい。
- 計画時にどのクラスを変更しますか？
- Solver で変更する変数はどのクラスにありますか？

そのクラスが、プランニングエンティティです。

プランニングエンティティークラスは、**@PlanningEntity** アノテーションでアノテートする必要があります。または、ドメインデザイナーで Red Hat ビルドの OptaPlanner ドックを使用して Business Central に定義する必要があります。

各プランニングエンティティークラスには、1つ以上の **プランニング変数** があり、1つ以上の定義プロパティが必要です。

多くのユースケースには、プランニングエンティティークラスが1つだけあり、1つのプランニングエンティティークラスに対してプランニング変数が1つだけ含まれます。

### 5.2.3. 従業員の勤務表プロジェクトへのデータモデルの作成

このセクションでは、Business Central で従業員の勤務表サンプルプロジェクトを実行するのに必要なデータオブジェクトを作成します。

#### 前提条件

- 「[従業員の勤務表プロジェクトの設定](#)」に従ってプロジェクト設定が完了している。

#### 手順

- 新規プロジェクトで、プロジェクトパースペクティブの **Data Object** をクリックするか、**Add Asset → Data Object** をクリックして、新しいデータオブジェクトを作成します。
- 最初のデータオブジェクトの名前を **Timeslot** とし、**パッケージ** で **employee rostering.employee rostering** を選択します。  
OK をクリックします。
- Data Objects** パースペクティブで **+add field** をクリックして、**Timeslot** データオブジェクトにフィールドを追加します。
- id** フィールドで **endTime** と入力します。
- Type** の横にあるドロップダウンメニューをクリックし、**LocalDateTime** を選択します。
- Create and continue** を別のフィールドに追加します。
- id** **startTime** および **Type** **LocalDateTime** を使用して、フィールドを追加します。
- Create** をクリックします。
- 右上の **Save** をクリックして、**Timeslot** データオブジェクトを保存します。
- 右上の **x** をクリックして、**Data Objects** パースペクティブを閉じ、**Assets** メニューに戻ります。
- 前述の手順で、以下のデータオブジェクトとその属性を作成します。

表5.1 Skill

id	タイプ
name	String

表5.2 Employee

id	タイプ
name	String
skills	employee rostering.employee rostering.Skill[List]

表5.3 Shift

id	タイプ
requiredSkill	employee rostering.employee rostering.Skill
timeslot	employee rostering.employee rostering.Timeslot

表5.4 DayOffRequest

id	タイプ
date	LocalDate
employee	employee rostering.employee rostering.Employee

表5.5 ShiftAssignment

id	タイプ
employee	employee rostering.employee rostering.Employee
shift	employee rostering.employee rostering.Shift

データオブジェクトの作成例は [デシジョンサービスの使用ガイド](#) を参照してください。

### 5.2.3.1. 従業員の勤務表プランニングエンティティの作成

従業員勤務表の計画問題を解決するには、プランニングエンティティと Solver を作成する必要があります。プランニングエンティティは、Red Hat ビルドの OptaPlanner ドックで利用可能な属性を使用して、ドメインデザイナーに定義します。

以下の手順に従って、従業員の勤務表サンプルに、**ShiftAssignment** データオブジェクトをプランニングエンティティとして定義します。

## 前提条件

- 従業員の勤務表サンプルを実行するには、「[従業員の勤務表プロジェクトへのデータモデルの作成](#)」の手順に従って、関連するデータオブジェクトとプランニングエンティティを作成する必要があります。

## 手順

- プロジェクトの **Assets** メニューから、**ShiftAssignment** データオブジェクトを開きます。
- Data Objects** パースペクティブで、右側の  をクリックして、OptaPlanner のドックを開きます。
- Planning Entity** を選択します。
- ShiftAssignment** データオブジェクトのフィールドリストで **employee** を選択します。
- OptaPlanner ドックで **Planning Variable** を選択します。  
**Value Range Id** 入力フィールドに **employeeRange** を入力します。これにより、**@ValueRangeProvider** アノテーションがプランニングエンティティに追加され、デザイナーの **Source** タブをクリックすると表示されます。  
 プランニング変数の値の範囲は **@ValueRangeProvider** アノテーションで定義されます。**@ValueRangeProvider** アノテーションには **id** プロパティが常にあり、**@PlanningVariable** の **valueRangeProviderRefs** プロパティから参照されます。
- ドックを閉じ、**Save** をクリックして、データオブジェクトを保存します。

### 5.2.3.2. 従業員の勤務表プランニングソリューションの作成

従業員勤務表の問題は、定義したプランニングソリューションに依存します。プランニングソリューションは、Red Hat ビルドの OptaPlanner ドックで利用可能な属性を使用して、ドメインデザイナーに定義します。

## 前提条件

- 「[従業員の勤務表プロジェクトへのデータモデルの作成](#)」および「[従業員の勤務表プランニングエンティティの作成](#)」の手順に従って、従業員の勤務表サンプルを実行するのに必要なデータオブジェクトおよびプランニングエンティティを作成している。

## 手順

- 識別子 **EmployeeRoster** でデータオブジェクトを新規作成します。
- 以下のフィールドを作成します。

表5.6 EmployeeRoster

id	タイプ
dayOffRequestList	employee rostering.employee rostering.DayOffRequest[List]



id	タイプ
shiftAssignmentList	employee rostering.employee rostering.ShiftAssignment[List]
shiftList	employee rostering.employee rostering.Shift[List]
skillList	employee rostering.employee rostering.Skill[List]
timeslotList	employee rostering.employee rostering.Timeslot[List]

3. **Data Objects** パースペクティブで、右側の  をクリックして、OptaPlanner のドックを開きます。
4. **Planning Solution** を選択します。
5. **Solution Score Type** は、デフォルトの **Hard soft score** のままにします。これにより、タイプがソリューションスコアとなる **EmployeeRoster** データオブジェクトに、**score** フィールドが自動的に生成されます。
6. 次の属性で新しいフィールドを追加します。

id	タイプ
employeeList	employee rostering.employee rostering.Employee[List]

7. **employeeList** フィールドを選択した状態で、Red Hat ビルドの OptaPlanner ドックを開いて、**Planning Value Range Provider** ボックスを選択します。  
id フィールドに **employeeRange** を入力します。ドックを閉じます。
8. 右上で **Save** をクリックし、アセットを保存します。

#### 5.2.4. 従業員勤務表の制約

従業員の勤務表はプランニングソリューションです。すべての計画問題には、最適解を得るのに満たさなければならない制約が含まれます。

Business Central の従業員の勤務表サンプルプロジェクトには、以下のハード制約およびソフト制約が含まれます。

##### ハード制約

- 従業員に割り当てられるシフトの数は、1日1つまで。
- 特別な従業員スキルが必要なすべてのシフトは、そのスキルを持つ従業員に割り当てられる。

## ソフト制約

- すべての従業員がシフトに割り当てられている。
- 従業員が休暇を取った場合は、シフトが別の従業員に再割り当てされる。

ハード制約およびソフト制約は、Free form DRL デザイナー、またはガイド付きルールを使用して Business Central で定義します。

### 5.2.4.1. DRL (Drools Rule Language) ルール

DRL (Drools Rule Language) ルールは、**.drl** テキストファイルに直接定義するビジネスルールです。このような DRL ファイルは、Business Central の他のすべてのルールアセットが最終的にレンダリングされるソースとなります。Business Central インターフェイスで DRL ファイルを作成して管理するか、Red Hat CodeReady Studio や別の統合開発環境 (IDE) を使用して Maven または Java プロジェクトの一部として外部で作成することができます。DRL ファイルには、最低でもルールの条件 (**when**) およびアクション (**then**) を定義するルールを 1 つ以上追加できます。Business Central の DRL デザイナーでは、Java、DRL、および XML の構文が強調表示されます。

DRL ファイルは、以下のコンポーネントで設定されます。

#### DRL ファイル内のコンポーネント

```
package

import

function // Optional

query // Optional

declare // Optional

global // Optional

rule "rule name"
  // Attributes
  when
    // Conditions
  then
    // Actions
end

rule "rule2 name"

...
```

以下の DRL ルールの例では、ローン申し込みのデシジョンサービスで年齢制限を指定します。

#### 申込者の年齢制限に関するルールの例

```
rule "Underage"
  salience 15
  agenda-group "applicationGroup"
```

```

when
  $application : LoanApplication()
  Applicant( age < 21 )
then
  $application.setApproved( false );
  $application.setExplanation( "Underage" );
end

```

DRL ファイルには、ルール、クエリー、関数が1つまたは複数含まれており、このファイルで、ルールやクエリーで割り当て、使用するインポート、グローバル、属性などのリソース宣言を定義できます。DRL パッケージは、DRL ファイルの一番上に表示され、ルールは通常最後に表示されます。他の DRL コンポーネントはどのような順番でも構いません。

ルールごとに、ルールパッケージ内で一意の名前を指定する必要があります。パッケージ内の DRL ファイルで、同じルール名を複数回使用すると、ルールのコンパイルに失敗します。特にルール名にスペースを使用する場合など、ルール名には必ず二重引用符 (**rule "rule name"**) を使用して、コンパイルエラーが発生しないようにしてください。

DRL ルールに関連するデータオブジェクトはすべて、DRL ファイルと同じ Business Central プロジェクトパッケージに置く必要があります。同じパッケージに含まれるアセットはデフォルトでインポートされます。その他のパッケージの既存アセットは、DRL ルールを使用してインポートできます。

#### 5.2.4.2. DRL デザイナーを使用した従業員の勤務表の制約定義

Business Central で Free form DRL デザイナーを使用して、従業員の勤務表サンプルに制約の定義を作成できます。

この手順を使用して、シフトが終わってから 10 時間以上経たないと従業員をシフトに割り当てられない **ハード制約** を作成します。

##### 手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → DRL ファイル** の順にクリックします。
3. **DRL file** 名前フィールドに、**ComplexScoreRules** と入力します。
4. **employeeerostering.employeeerostering** パッケージを選択します。
5. **+OK** をクリックして DRL ファイルを作成します。
6. DRL デザイナーの **Model** タブで、**Employee10HourShiftSpace** ルールを DRL ファイルとして定義します。

```

package employeeerostering.employeeerostering;

rule "Employee10HourShiftSpace"
  when
    $shiftAssignment : ShiftAssignment( $employee : employee != null, $shiftEndDateTime :
shift.timeslot.endTime)
    ShiftAssignment( this != $shiftAssignment, $employee == employee, $shiftEndDateTime
<= shift.timeslot.endTime,
    $shiftEndDateTime.until(shift.timeslot.startTime,
java.time.temporal.ChronoUnit.HOURS) <10)
  then

```

```

    then
        scoreHolder.addHardConstraintMatch(kcontext, -1);
    end

```

7. **Save** をクリックして、DRL ファイルを保存します。

DRL ファイルの作成方法は [DRL ルールを使用したデシジョンサービスの作成](#) を参照してください。

## 5.2.5. ガイド付きルールを使用して従業員の勤務表にルールの作成

Business Central でガイド付きルールデザイナーを使用して、従業員の勤務表にハード制約およびソフト制約を定義するルールを作成できます。

### 5.2.5.1. ガイド付きルール

ガイド付きルールは、ルール作成のプロセスを提供する、Business Central の UI ベースのガイド付きルールデザイナーで作成するビジネスルールです。ガイド付きルールデザイナーを使用すると、ルールを定義するデータオブジェクトに基づいて、可能なインプットにフィールドおよびオプションを提供します。定義したガイド付きルールは、その他のすべてのルールアセットとともに Drools Rule Language (DRL) ルールにコンパイルされます。




ガイド付きルールに関連するすべてのデータオブジェクトは、ガイド付きルールと同じプロジェクトパッケージに置く必要があります。同じパッケージに含まれるアセットはデフォルトでインポートされます。必要なデータオブジェクトとガイド付きルールを作成したら、ガイド付きルールデザイナーの **Data Objects** タブから、必要なデータオブジェクトがすべてリストされていることを検証したり、**新規アイテム** を追加してその他の既存データオブジェクトをインポートしたりできます。

### 5.2.5.2. 従業員のシフト数のバランスを取るガイド付きルールの作成

ガイド付きルール **BalanceEmployeesShiftNumber** は、可能な限りバランスを取るように従業員にシフトを割り当てるソフト制約を作成します。これは、シフトの分配が平等でなくなると増えるスコアペナルティーを作成することで行います。ルールによって実装されたスコア式により、Solver がよりバランスの取れるようにシフトを分散させます。

## 手順

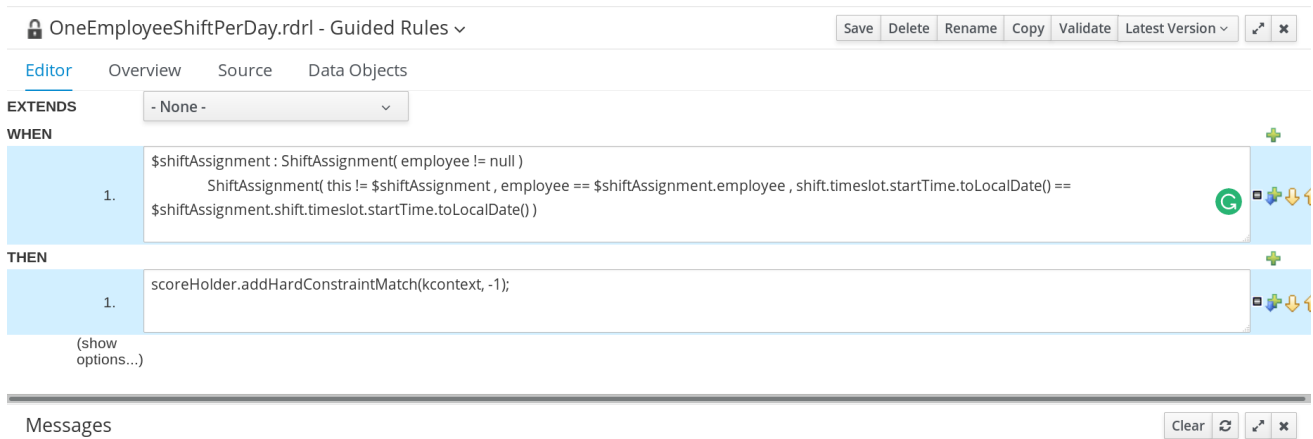
1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Guided Rule** の順にクリックします。

3. Guided Rule 名に **BalanceEmployeesShiftNumber** と入力し、Package で **employeeerostring.employeeerostring** を選択します。
4. OK をクリックして、ルールアセットを作成します。
5. WHEN フィールドで  をクリックして、WHEN 条件を追加します。
6. Add a condition to the rule ウィンドウで **Employee** を選択します。+OK をクリックします。
7. **Employee** 条件でクリックして制約を修正し、変数名 **\$employee** を追加します。
8. WHEN 条件 **From Accumulate** を追加します。
  - a. **From Accumulate** 条件の上で click to add pattern をクリックし、ドロップダウンリストでファクトタイプ **Number** を選択します。
  - b. 変数名 **\$shiftCount** を **Number** 条件に追加します。
  - c. **From Accumulate** 条件の下で click to add pattern をクリックして、ドロップダウンリストで **ShiftAssignment** ファクトタイプを選択します。
  - d. 変数名 **\$shiftAssignment** を **ShiftAssignment** ファクトタイプに追加します。
  - e. **ShiftAssignment** 条件を再度クリックし、Add a restriction on a field ドロップダウンリストで **employee** を選択します。
  - f. **employee** 制約の横にあるドロップダウンリストで **equal to** を選択します。
  - g. ドロップダウンボタンの横の  アイコンをクリックして変数を追加し、Field value ウィンドウで **Bound variable** をクリックします。
  - h. ドロップダウンリストで **\$employee** を選択します。
  - i. Function ボックスに **count (\$shiftAssignment)** と入力します。
9. THEN フィールドで  をクリックして、THEN 条件を追加します。
10. Add a new action ウィンドウで **Modify Soft Score** を選択します。+OK をクリックします。
  - a. ボックスに **- (\$shiftCount.intValue()\*\$shiftCount.intValue())** と入力します。
11. 右上隅の **Validate** をクリックし、ルール条件がすべて有効であることを確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルールの全コンポーネントを見直し、エラーが表示されなくなるまでルールの妥当性確認を行います。
12. **Save** をクリックして、ルールを保存します。


ガイド付きルールの作成方法は [ガイド付きルールを使用したデジモンサービスの作成](#) を参照してください。

### 5.2.5.3. 同じ日に複数のシフトを割り当てないようにするガイド付きルールの作成

ガイド付きルール **OneEmployeeShiftPerDay** は、同じ日の複数のシフトに従業員を割り当てないようにするハード制約を作成します。従業員の勤務表サンプルでは、この制約はガイド付きルールデザイナーを使用して作成されます。




## 手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Guided Rule** の順にクリックします。
3. **Guided Rule** 名に **OneEmployeeShiftPerDay** と入力し、**Package** で **employee rostering.employee rostering** を選択します。
4. **OK** をクリックして、ルールアセットを作成します。
5. **WHEN** フィールドで  をクリックして、**WHEN** 条件を追加します。
6. **Add a condition to the rule** ウィンドウから **Free form DRL** を選択します。
7. Free form の DRL ボックスに、以下の条件を入力します。

```
$shiftAssignment : ShiftAssignment( employee != null )
    ShiftAssignment( this != $shiftAssignment , employee == $shiftAssignment.employee ,
    shift.timeslot.startTime.toLocalDate() ==
    $shiftAssignment.shift.timeslot.startTime.toLocalDate() )
```

この条件は、同じ日に別のシフトがすでに割り当てられている従業員にはシフトを割り当てることができないことを示しています。

8. **THEN** フィールドで  をクリックして、**THEN** 条件を追加します。
  9. **Add a new action** ウィンドウから **Add Free form DRL** を選択します。
  10. Free form の DRL ボックスに、以下の条件を入力します。
- ```
scoreHolder.addHardConstraintMatch(kcontext, -1);
```
11. 右上隅の **Validate** をクリックし、ルール条件がすべて有効であることを確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルールの全コンポーネントを見直し、エラーが表示されなくなるまでルールの妥当性確認を行います。
  12. **Save** をクリックして、ルールを保存します。

ガイド付きルールの作成方法は [ガイド付きルールを使用したデシジョンサービスの作成](#) を参照してください。

#### 5.2.5.4. シフト要件にスキルを一致させるガイド付きルールを作成

ガイド付きルール **ShiftRequiredSkillsAreMet** は、すべてのシフトが、適切なスキルセットを持つ従業員に割り当てられるのを確認するハード制約を作成します。従業員の勤務表サンプルでは、この制約はガイド付きルールデザイナーを使用して作成されます。

ShiftRequiredSkillsAreMet.rdl - Guided Rules ▾

Save Delete Rename Copy Validate Latest Version ▾

Editor Overview Source Data Objects

EXTENDS - None - ▾

WHEN

1. There is a ShiftAssignment with:

employee is not null

[**\$requiredSkill**] :shift.requiredSkill, Choose... --- please choose ---

[not bound]:employee.skills, Choose... excludes \$requiredSkill

THEN



1. Hard Score -1

(show options...)

Messages Clear

#### 手順

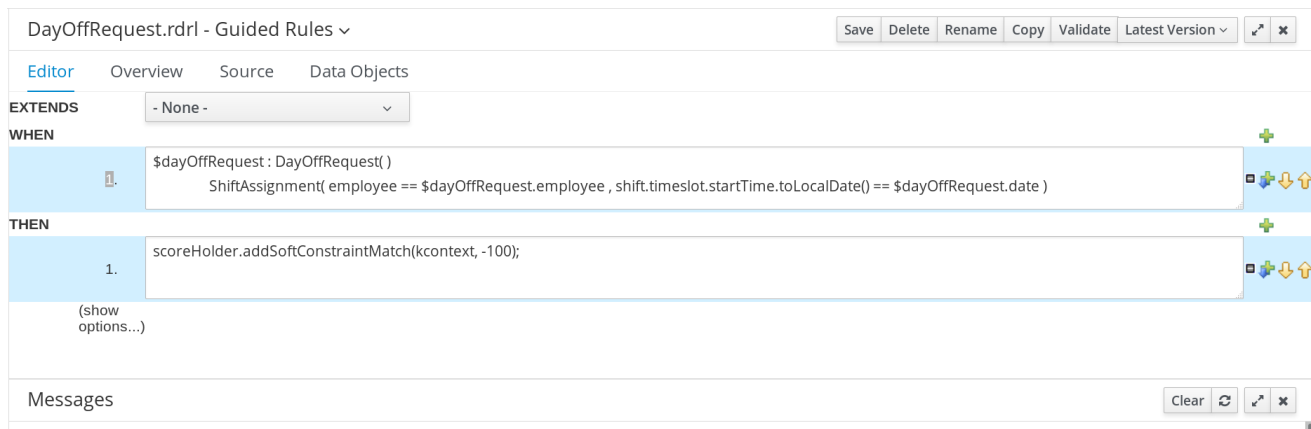
1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Guided Rule** の順にクリックします。
3. **Guided Rule** 名に **ShiftRequiredSkillsAreMet** と入力し、**Package** で **employee rostering.employee rostering** を選択します。
4. **OK** をクリックして、ルールアセットを作成します。
5. **WHEN** フィールドで **+** をクリックして、**WHEN** 条件を追加します。
6. **Add a condition to the rule** ウィンドウで **ShiftAssignment** を選択します。 **+OK** をクリックします。
7. **ShiftAssignment** 条件をクリックし、**Add a restriction on a field** ドロップダウンリストで **employee** を選択します。
8. デザイナーで、**employee** の横のドロップダウンリストをクリックし、**is not null** を選択します。
9. **ShiftAssignment** 条件をクリックし、**Expression editor** をクリックします。
  - a. デザイナーで、**[not bound]** をクリックし、**Expression editor** を開き、式と変数 **\$requiredSkill** をバインドします。 **Set** をクリックします。
  - b. デザイナーの **\$requiredSkill** の横にあるドロップダウンリストで **shift** を選択し、その隣のドロップダウンリストで **requiredSkill** を選択します。
10. **ShiftAssignment** 条件をクリックし、**Expression editor** をクリックします。
  - a. デザイナーで、**[not bound]** の横にあるドロップダウンリストで **employee** を選択し、その隣のドロップダウンリストで **skills** を選択します。
  - b. その隣のドロップダウンリストでは **Choose** を選択したままにします。

- c. その隣のドロップダウンボックスで、**please choose** を **excludes** に変更します。
- d. **excludes** の横にある  アイコンをクリックし、Field value ウィンドウで **New formula** ボタンをクリックします。
- e. 式ボックスに **\$requiredSkill** を追加します。
11. **THEN** フィールドで  をクリックして、**THEN** 条件を追加します。
12. **Add a new action** ウィンドウで **Modify Hard Score** を選択します。 +OK をクリックします。
13. スコアアクションボックスに **-1** を入力します。
14. 右上隅の **Validate** をクリックし、ルール条件がすべて有効であることを確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルールの全コンポーネントを見直し、エラーが表示されなくなるまでルールの妥当性確認を行います。
15. **Save** をクリックして、ルールを保存します。

ガイド付きルールの作成方法は [ガイド付きルールを使用したデシジョンサービスの作成](#) を参照してください。

#### 5.2.5.5. 休暇申請を管理するガイド付きルールの作成

ガイド付きルール **DayOffRequest** は、ソフト制約を作成します。この制約では、そのシフトに元々割り当てられていた従業員がその日に就業できなくなった場合に、別の従業員にシフトを再割り当てできるようにできます。従業員の勤務表サンプルでは、この制約はガイド付きルールデザイナーを使用して作成されます。



#### 手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Guided Rule** の順にクリックします。
3. **Guided Rule** 名に **DayOffRequest** と入力し、**Package** で **employee rostering.employee rostering** を選択します。
4. **OK** をクリックして、ルールアセットを作成します。
5. **WHEN** フィールドで  をクリックして、**WHEN** 条件を追加します。




6. **Add a condition to the rule** ウィンドウから **Free form DRL** を選択します。

7. Free form の DRL ボックスに、以下の条件を入力します。

```
$dayOffRequest : DayOffRequest( )
    ShiftAssignment( employee == $dayOffRequest.employee ,
    shift.timeslot.startTime.toLocalDate() == $dayOffRequest.date )
```

この条件は、休暇申請を行った従業員にシフトを割り当てている場合に、その従業員をその日のシフト割り当てから削除できることを示しています。

8. **THEN** フィールドで  をクリックして、**THEN** 条件を追加します。

9. **Add a new action** ウィンドウから **Add Free form DRL** を選択します。

10. Free form の DRL ボックスに、以下の条件を入力します。

```
scoreHolder.addSoftConstraintMatch(kcontext, -100);
```

11. 右上隅の **Validate** をクリックし、ルール条件がすべて有効であることを確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルールの全コンポーネントを見直し、エラーが表示されなくなるまでルールの妥当性確認を行います。

12. **Save** をクリックして、ルールを保存します。

ガイド付きルールの作成方法は [ガイド付きルールを使用したデシジョンサービスの作成](#) を参照してください。

### 5.2.6. 従業員の勤務表の Solver 設定の作成

Business Central に Solver 設定を作成して編集できます。Solver 設定デザイナーは、プロジェクトがデプロイされた後に実行できる Solver 設定を作成します。

#### 前提条件

- Red Hat Decision Manager をダウンロードしてインストールしている。
- 従業員の勤務表サンプルに関連するアセットをすべて作成して設定している。

#### 手順

- Business Central で **Menu → Projects** をクリックし、使用するプロジェクトをクリックして開きます。
- Assets** パースペクティブで、**Add Asset → Solver configuration** をクリックします。
- Create new Solver configuration** ウィンドウで、Solver の名前 **EmployeeRosteringSolverConfig** と入力し、**Ok** をクリックします。  
これにより、**Solver configuration** デザイナーが開きます。
- Score Director Factory** 設定セクションで、スコアリングルール定義を含む KIE ベースを定義します。従業員の勤務表サンプルプロジェクトは **defaultKieBase** を使用します。
  - KIE ベースに定義した KIE セッションの中から1つ選択します。従業員の勤務表サンプルプロジェクトは **defaultKieSession** を使用します。

5. 右上の **Validate** をクリックし、**Score Director Factory** 設定が正しいことを確認します。妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、エラーが表示されなくなるまで妥当性確認を行います。
6. **Save** をクリックして、Solver 設定を保存します。

### 5.2.7. 従業員の勤務表プロジェクトに対する Solver の終了設定

一定期間が過ぎたら Solver が終了できるように設定できます。デフォルトでは、プランニングエンジンには、時間制限なく問題を解決できるように指定されています。

従業員の勤務表サンプルプロジェクトは、30 秒間実行するように設定されています。

#### 前提条件

- 従業員の勤務表プロジェクトに、関連するすべてのアセットを作成し、[「従業員の勤務表の Solver 設定の作成」](#) の手順に従って、Business Central に Solver 設定 **EmployeeRosteringSolverConfig** を作成している。

#### 手順

1. **Assets** パースペクティブで **EmployeeRosteringSolverConfig** を開きます。これにより、**Solver 設定 デザイナー**が開きます。
2. **Termination** セクションで **Add** をクリックして、選択した論理グループに新しい終了要素を作成します。
3. ドロップダウンリストから、終了タイプ **Time spent** を選択します。これは、終了条件の入力フィールドとして追加されます。
4. 時間要素の横の矢印を使用して、経過時間を 30 秒に設定します。
5. 右上の **Validate** をクリックし、**Score Director Factory** 設定が正しいことを確認します。妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、エラーが表示されなくなるまで妥当性確認を行います。
6. **Save** をクリックして、Solver 設定を保存します。

## 5.3. REST API を使用した SOLVER へのアクセス

サンプルの Solver をデプロイするか再作成したら、REST API を使用してアクセスできます。

REST API を使用して Solver インスタンスを登録する必要があります。その後に、データセットを指定して、最適解を取得できます。

#### 前提条件

- 本書の以前のセクションに従い、従業員の勤務表プロジェクトを設定し、デプロイしてください。[「従業員の勤務表サンプルプロジェクトの再作成」](#) に記載のとおり、同じプロジェクトをデプロイするか、[「Business Central への従業員勤務表サンプルプロジェクトのデプロイメント」](#) に記載のとおり、プロジェクトを再作成できます。

### 5.3.1. REST API を使用した Solver の登録

Solver を使用するには、REST API を使用して Solver インスタンスを登録する必要があります。

各 Solver インスタンスで、一度に最適化できる計画問題の数は1つだけです。

## 手順

1. 以下のヘッダーを使用して HTTP 要求を作成します。

```
authorization: admin:admin
X-KIE-ContentType: xstream
content-type: application/xml
```

2. 以下の要求を使用して Solver を登録します。

### PUT

**http://localhost:8080/kie-server/services/rest/server/containers/employee rostering\_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver**

#### 要求ボディー

```
<solver-instance>
  <solver-config-
file>employee rostering/employee rostering/EmployeeRosteringSolverConfig.solver.xml</s
olver-config-file>
</solver-instance>
```

## 5.3.2. REST API を使用した Solver の呼び出し

Solver インスタンスを登録した後に、REST API を使用して、データセットを Solver に送信して、最適解を取得できます。

## 手順

1. 以下のヘッダーを使用して HTTP 要求を作成します。

```
authorization: admin:admin
X-KIE-ContentType: xstream
content-type: application/xml
```

2. 以下の例のように、データセットを含む Solver に要求を送信します。

### POST

**http://localhost:8080/kie-server/services/rest/server/containers/employee rostering\_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver/state/solving**

#### 要求ボディー

```
<employee rostering.employee rostering.EmployeeRoster>
  <employeeList>
    <employee rostering.employee rostering.Employee>
      <name>John</name>
      <skills>
        <employee rostering.employee rostering.Skill>
          <name>reading</name>
```

```

    </employee rostering.employee rostering.Skill>
  </skills>
</employee rostering.employee rostering.Employee>
<employee rostering.employee rostering.Employee>
  <name>Mary</name>
  <skills>
    <employee rostering.employee rostering.Skill>
      <name>writing</name>
    </employee rostering.employee rostering.Skill>
  </skills>
</employee rostering.employee rostering.Employee>
<employee rostering.employee rostering.Employee>
  <name>Petr</name>
  <skills>
    <employee rostering.employee rostering.Skill>
      <name>speaking</name>
    </employee rostering.employee rostering.Skill>
  </skills>
</employee rostering.employee rostering.Employee>
</employeeList>
<shiftList>
  <employee rostering.employee rostering.Shift>
    <timeslot>
      <startTime>2017-01-01T00:00:00</startTime>
      <endTime>2017-01-01T01:00:00</endTime>
    </timeslot>
    <requiredSkill
reference="../../employeeList/employee rostering.employee rostering.Employee/skills/employee rostering.employee rostering.Skill"/>
    </employee rostering.employee rostering.Shift>
  <employee rostering.employee rostering.Shift>
    <timeslot reference="../../employee rostering.employee rostering.Shift/timeslot"/>
    <requiredSkill
reference="../../employeeList/employee rostering.employee rostering.Employee[3]/skills/employee rostering.employee rostering.Skill"/>
    </employee rostering.employee rostering.Shift>
  <employee rostering.employee rostering.Shift>
    <timeslot reference="../../employee rostering.employee rostering.Shift/timeslot"/>
    <requiredSkill
reference="../../employeeList/employee rostering.employee rostering.Employee[2]/skills/employee rostering.employee rostering.Skill"/>
    </employee rostering.employee rostering.Shift>
  </shiftList>
<skillList>
  <employee rostering.employee rostering.Skill
reference="../../employeeList/employee rostering.employee rostering.Employee/skills/employee rostering.employee rostering.Skill"/>
  <employee rostering.employee rostering.Skill
reference="../../employeeList/employee rostering.employee rostering.Employee[3]/skills/employee rostering.employee rostering.Skill"/>
  <employee rostering.employee rostering.Skill
reference="../../employeeList/employee rostering.employee rostering.Employee[2]/skills/employee rostering.employee rostering.Skill"/>
</skillList>
<timeslotList>
  <employee rostering.employee rostering.Timeslot

```

```

reference="../../shiftList/employee rostering.employee rostering.Shift/timeslot"/>
</timeslotList>
<dayOffRequestList/>
<shiftAssignmentList>
  <employee rostering.employee rostering.ShiftAssignment>
    <shift reference="../../shiftList/employee rostering.employee rostering.Shift"/>
  </employee rostering.employee rostering.ShiftAssignment>
  <employee rostering.employee rostering.ShiftAssignment>
    <shift reference="../../shiftList/employee rostering.employee rostering.Shift[3]"/>
  </employee rostering.employee rostering.ShiftAssignment>
  <employee rostering.employee rostering.ShiftAssignment>
    <shift reference="../../shiftList/employee rostering.employee rostering.Shift[2]"/>
  </employee rostering.employee rostering.ShiftAssignment>
</shiftAssignmentList>
</employee rostering.employee rostering.EmployeeRoster>

```

3. 計画問題に最適解をリクエストします。

GET

**http://localhost:8080/kie-  
server/services/rest/server/containers/employee rostering\_1.0.0-  
SNAPSHOT/solvers/EmployeeRosteringSolver/bestsolution**

応答例

```

<solver-instance>
  <container-id>employee-rostering</container-id>
  <solver-id>solver1</solver-id>
  <solver-config-
file>employee rostering/employee rostering/EmployeeRosteringSolverConfig.solver.xml</s
olver-config-file>
  <status>NOT_SOLVING</status>
  <score
scoreClass="org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore">0hard/0soft<
/score>
  <best-solution class="employee rostering.employee rostering.EmployeeRoster">
    <employeeList>
      <employee rostering.employee rostering.Employee>
        <name>John</name>
        <skills>
          <employee rostering.employee rostering.Skill>
            <name>reading</name>
          </employee rostering.employee rostering.Skill>
        </skills>
      </employee rostering.employee rostering.Employee>
      <employee rostering.employee rostering.Employee>
        <name>Mary</name>
        <skills>
          <employee rostering.employee rostering.Skill>
            <name>writing</name>
          </employee rostering.employee rostering.Skill>
        </skills>
      </employee rostering.employee rostering.Employee>
      <employee rostering.employee rostering.Employee>
        <name>Petr</name>

```

```

    <skills>
      <employee rostering.employee rostering.Skill>
        <name>speaking</name>
      </employee rostering.employee rostering.Skill>
    </skills>
  </employee rostering.employee rostering.Employee>
</employeeList>
<shiftList>
  <employee rostering.employee rostering.Shift>
    <timeslot>
      <startTime>2017-01-01T00:00:00</startTime>
      <endTime>2017-01-01T01:00:00</endTime>
    </timeslot>
    <requiredSkill
reference="../../../../employeeList/employee rostering.employee rostering.Employee/skills/employee rostering.employee rostering.Skill"/>
    </employee rostering.employee rostering.Shift>
  </employee rostering.employee rostering.Shift>
    <timeslot reference="../../employee rostering.employee rostering.Shift/timeslot"/>
    <requiredSkill
reference="../../../../employeeList/employee rostering.employee rostering.Employee[3]/skills/employee rostering.employee rostering.Skill"/>
    </employee rostering.employee rostering.Shift>
  </employee rostering.employee rostering.Shift>
    <timeslot reference="../../employee rostering.employee rostering.Shift/timeslot"/>
    <requiredSkill
reference="../../../../employeeList/employee rostering.employee rostering.Employee[2]/skills/employee rostering.employee rostering.Skill"/>
    </employee rostering.employee rostering.Shift>
</shiftList>
<skillList>
  <employee rostering.employee rostering.Skill
reference="../../../../employeeList/employee rostering.employee rostering.Employee/skills/employee rostering.employee rostering.Skill"/>
  <employee rostering.employee rostering.Skill
reference="../../../../employeeList/employee rostering.employee rostering.Employee[3]/skills/employee rostering.employee rostering.Skill"/>
  <employee rostering.employee rostering.Skill
reference="../../../../employeeList/employee rostering.employee rostering.Employee[2]/skills/employee rostering.employee rostering.Skill"/>
</skillList>
<timeslotList>
  <employee rostering.employee rostering.Timeslot
reference="../../shiftList/employee rostering.employee rostering.Shift/timeslot"/>
</timeslotList>
<dayOffRequestList/>
<shiftAssignmentList/>
<score>0hard/0soft</score>
</best-solution>
</solver-instance>

```

## 第6章 OPTAPLANNER および QUARKUS の使用ガイド

<https://code.quarkus.redhat.com> の Web サイトを使用して Red Hat ビルドの OptaPlanner Quarkus の Maven プロジェクトを生成し、アプリケーションで使用する拡張機能を自動的に追加および設定できます。その後、Quarkus Maven リポジトリのダウンロードや、プロジェクトでのオンライン Maven リポジトリの使用が可能になります。

### 6.1. APACHE MAVEN および RED HAT ビルドの QUARKUS

Apache Maven は分散型構築自動化ツールで、ソフトウェアプロジェクトの作成、ビルド、および管理を行うために Java アプリケーション開発で使用されます。Maven は Project Object Model (POM) ファイルと呼ばれる標準の設定ファイルを使用して、プロジェクトの定義や構築プロセスの管理を行います。POM ファイルは、モジュールおよびコンポーネントの依存関係、ビルドの順番、結果となるプロジェクトパッケージのターゲットを記述し、XML ファイルを使用して出力します。これにより、プロジェクトが適切かつ統一された状態でビルドされるようになります。

#### Maven リポジトリ

Maven リポジトリには、Java ライブラリー、プラグイン、およびその他のビルドアーティファクトが格納されます。デフォルトのパブリックリポジトリは Maven 2 Central Repository ですが、複数の開発チームの間で共通のアーティファクトを共有する目的で、社内のプライベートおよび内部リポジトリとすることが可能です。また、サードパーティーのリポジトリも利用できます。

Quarkus プロジェクトでオンライン Maven リポジトリを使用するか、Red Hat ビルドの Quarkus の Maven リポジトリをダウンロードできます。

#### Maven プラグイン

Maven プラグインは、POM ファイルの定義済みの部分で1つ以上のゴールを達成します。Quarkus アプリケーションは以下の Maven プラグインを使用します。

- Quarkus Maven プラグイン (**quarkus-maven-plugin**): Maven による Quarkus プロジェクトの作成を実現して、uber-JAR ファイルの生成をサポートし、開発モードを提供します。
- Maven Surefire プラグイン (**maven-surefire-plugin**): ビルドライフサイクルのテストフェーズで使用され、アプリケーションでユニットテストを実行します。プラグインは、テストレポートが含まれるテキストファイルと XML ファイルを生成します。

#### 6.1.1. オンラインリポジトリの Maven の **settings.xml** ファイルの設定

ユーザーの **settings.xml** ファイルを設定して、Maven プロジェクトでオンライン Maven リポジトリを使用できます。これは、推奨の手法です。リポジトリマネージャーまたは共有サーバー上のリポジトリと使用する Maven 設定は、プロジェクトの制御および管理性を向上させます。



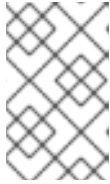
#### 注記

Maven の **settings.xml** ファイルを変更してリポジトリを設定する場合、変更はすべての Maven プロジェクトに適用されます。

#### 手順

1. テキストエディターまたは統合開発環境 (IDE) で Maven の `~/.m2/settings.xml` ファイルを開きます。





### 注記

~/m2/ ディレクトリーに **settings.xml** ファイルがない場合には、**\$MAVEN\_HOME/m2/conf/** ディレクトリーから ~/m2/ ディレクトリーに **settings.xml** ファイルをコピーします。

- 以下の行を Maven の **settings.xml** ファイルの **<profiles>** 要素に追加します。

```
<!-- Configure the Maven repository -->
<profile>
  <id>red-hat-enterprise-maven-repository</id>
  <repositories>
    <repository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

- 以下の行を **settings.xml** ファイルの **<activeProfiles>** 要素に追加し、ファイルを保存します。

```
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```

## 6.1.2. Quarkus Maven リポジトリーのダウンロードおよび設定

オンライン Maven リポジトリーを使用しない場合は、Quarkus Maven リポジトリーをダウンロードして設定できます。Quarkus Maven リポジトリーには、Java 開発者がアプリケーションの構築に使用する要件の多くが含まれています。この手順では、Maven の **settings.xml** ファイルを編集し、Quarkus Maven リポジトリーを設定する方法を説明します。



### 注記

Maven の **settings.xml** ファイルを変更してリポジトリーを設定する場合、変更はすべての Maven プロジェクトに適用されます。

### 手順



1. Red Hat カスタマーポータルの [Software Downloads](#) ページから Red Hat ビルドの Quarkus Maven リポジトリの ZIP ファイルをダウンロードします。
2. ダウンロードしたアーカイブを展開します。
3. `~/.m2/` ディレクトリに移動し、テキストエディターまたは統合開発環境 (IDE) で Maven の **settings.xml** ファイルを開きます。
4. 以下の行を **settings.xml** ファイルの **<profiles>** 要素に追加します。ここで、**QUARKUS\_MAVEN\_REPOSITORY** はダウンロードした Maven リポジトリのパスです。**QUARKUS\_MAVEN\_REPOSITORY** の形式は **file://\$PATH** でなければなりません。たとえば **file:///home/userX/rh-quarkus-2.2.3.GA-maven-repository/maven-repository** のようになります。

```
<!-- Configure the Quarkus Maven repository -->
<profile>
  <id>red-hat-enterprise-maven-repository</id>
  <repositories>
    <repository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>QUARKUS_MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>QUARKUS_MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

5. 以下の行を **settings.xml** ファイルの **<activeProfiles>** 要素に追加し、ファイルを保存します。

```
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```

## 重要

Maven リポジトリに古いアーティファクトが含まれる場合は、プロジェクトをビルドまたはデプロイしたときに以下のいずれかの Maven エラーメッセージが表示されることがあります。ここで、**ARTIFACT\_NAME** は不明なアーティファクトの名前で、**PROJECT\_NAME** は構築を試みているプロジェクトの名前になります。

- **Missing artifact PROJECT\_NAME**
- **[ERROR] Failed to execute goal on project ARTIFACT\_NAME; Could not resolve dependencies for PROJECT\_NAME**

この問題を解決するには、`~/.m2/repository` ディレクトリにあるローカルリポジトリのキャッシュバージョンを削除し、最新の Maven アーティファクトを強制的にダウンロードします。

## 6.2. MAVEN プラグインを使用した OPTAPLANNER RED HAT ビルドの QUARKUS MAVEN プロジェクトの作成

Apache Maven および Quarkus Maven プラグインを使用して、Red Hat ビルドの OptaPlanner および Quarkus アプリケーションの使用を開始できます。

### 前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスマーポータル (ログインが必要) の [ソフトウェアダウンロード](#) ページから入手できます。
- Apache Maven 3.6 以降がインストールされている。Maven は [Apache Maven Project](#) の Web サイトから入手できます。

### 手順

1. コマンドターミナルで以下のコマンドを入力し、Maven が JDK 11 を使用していること、そして Maven のバージョンが 3.6 以上であることを確認します。

```
mvn --version
```

2. 上記のコマンドで JDK 11 が返されない場合は、JDK 11 へのパスを PATH 環境変数に追加し、上記のコマンドを再度入力します。
3. Quarkus OptaPlanner クイックスタートプロジェクトを生成するには、以下のコマンドを入力します。

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:2.2.3.Final-redhat-00013:create \
  -DprojectId=com.example \
  -DprojectArtifactId=optaplanner-quickstart \
  -Dextensions="resteasy,resteasy-jackson,optaplanner-quarkus,optaplanner-quarkus-jackson" \
  -DplatformGroupId=com.redhat.quarkus.platform \
  -DplatformVersion=2.2.3.Final-redhat-00013 \
  -DnoExamples
```

このコマンドは、`./optaplanner-quickstart` ディレクトリで以下の要素を作成します。

- Maven の構造

- **src/main/docker** の **Dockerfile** ファイルの例
- アプリケーションの設定ファイル

表6.1 `mvn io.quarkus:quarkus-maven-plugin:2.2.3.Final-redhat-00013:create` コマンドで  
使用したプロパティ

プロパティ	説明
<b>projectGroupId</b>	プロジェクトのグループ ID。
<b>projectArtifactId</b>	プロジェクトのアーティファクト ID。
<b>extensions</b>	このプロジェクトで使用する Quarkus 拡張のコンマ区切りリスト。Quarkus 拡張の全一覧については、特定のコマンドラインで <b><code>mvn quarkus:list-extensions</code></b> を入力します。
<b>noExamples</b>	テストまたはクラスを使用せずに、プロジェクト構造でプロジェクトを作成します。

**projectGroupId** および **projectArtifactId** プロパティの値を使用して、プロジェクトバージョンを生成します。デフォルトのプロジェクトバージョンは **1.0.0-SNAPSHOT** です。

4. OptaPlanner プロジェクトを表示するには、OptaPlanner Quickstarts ディレクトリーに移動します。

```
cd optaplanner-quickstart
```

5. **pom.xml** ファイルを確認します。コンテンツの例を以下に示します。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.quarkus.platform</groupId>
      <artifactId>quarkus-bom</artifactId>
      <version>2.2.3.Final-redhat-00013</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>io.quarkus.platform</groupId>
      <artifactId>quarkus-optaplanner-bom</artifactId>
      <version>2.2.3.Final-redhat-00013</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy</artifactId>
```

```

</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-resteasy-jackson</artifactId>
</dependency>
<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-quarkus</artifactId>
</dependency>
<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-quarkus-jackson</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-junit5</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>

```

### 6.3. CODE.QUARKUS.REDHAT.COM を使用した RED HAT ビルドの QUARKUS MAVEN プロジェクトの作成

<https://code.quarkus.redhat.com> の Web サイトを使用して Red Hat ビルドの OptaPlanner Quarkus の Maven プロジェクトを生成し、アプリケーションで使用する拡張機能を自動的に追加および設定できます。さらに、[code.quarkus.redhat.com](https://code.quarkus.redhat.com) は、プロジェクトをネイティブ実行可能ファイルにコンパイルするために必要な設定パラメーターを自動的に管理します。

本セクションでは、以下のトピックを含む OptaPlanner Maven プロジェクトを生成するプロセスについて説明します。

- アプリケーションの基本情報を指定する
- プロジェクトに追加する拡張を選択する
- プロジェクトファイルでダウンロード可能なアーカイブを生成する
- アプリケーションのコンパイルおよび起動のカスタムコマンドを使用する

#### 前提条件

- Web ブラウザーがある。

#### 手順

1. Web ブラウザーで <https://code.quarkus.redhat.com> を開きます。
2. プロジェクトの詳細を指定します。
3. プロジェクトのグループ名を入力します。名前の形式は、Java パッケージ命名規則に従います (例: **com.example**)。
4. プロジェクトから生成された Maven アーティファクトに使用する名前を入力します (例: **code-with-quarkus**)。

5. **Build Tool > Maven** を選択して、作成するプロジェクトが Maven プロジェクトであることを指定します。選択するビルドツールにより、以下の項目が決定されます。

- 生成されたプロジェクトのディレクトリー構造。
- 生成されたプロジェクトで使用する設定ファイルの形式。
- アプリケーションのコンパイルおよび起動用のカスタムビルドスクリプトおよびコマンド (プロジェクトの生成後に **code.quarkus.redhat.com** が表示)。



#### 注記

Red Hat は、**code.quarkus.redhat.com** を使用した OptaPlanner Maven プロジェクトの作成だけをサポートします。Red Hat では、Gradle プロジェクトの生成はサポートしていません。

6. プロジェクトから生成されたアーティファクトで使用するバージョンを入力します。このフィールドのデフォルト値は **1.0.0-SNAPSHOT** です。semantic versioning の使用が推奨されますが、必要に応じて、別のタイプのバージョンを使用できます。
7. プロジェクトをパッケージ化する時に、ビルドツールが生成するアーティファクトのパッケージ名を入力します。  
パッケージ名は、Java パッケージの命名規則に従い、プロジェクトに使用するグループ名と一致するはずですが、別の名前を指定することもできます。



#### 注記

**code.quarkus.redhat.com** Web サイトは、OptaPlanner の最新リリースを自動的に使用します。プロジェクトの生成後に、**pom.xml** ファイルで BOM バージョンを手動で変更できます。

8. 以下の拡張を選択して、依存関係として組み込みます。

- RESTEasy JAX-RS (quarkus-resteasy)
  - RESTEasy Jackson (quarkus-resteasy-jackson)
  - OptaPlanner AI 制約ソルバー (optaplanner-quarkus)
  - OptaPlanner Jackson (optaplanner-quarkus-jackson)
- Red Hat は、一覧にある個別の拡張に対してさまざまなレベルのサポートを提供します。レベルは、各拡張名の横にあるラベルで示されています。
- **SUPPORTED** 拡張: Red Hat は、実稼働環境のエンタープライズアプリケーションでの使用を完全にサポートします。
  - **TECH-PREVIEW** 拡張: Red Hat は、[テクノロジープレビュー機能のサポート範囲](#) に基づき、限定的に、実稼働環境でのサポートを提供します。
  - **DEV-SUPPORT** 拡張: Red Hat は、実稼働環境での使用をサポートしていません。ただし、新規アプリケーションの開発での使用に対しては、Red Hat 開発者がこれらのコア機能をサポートしています。
  - **DEPRECATED** 拡張: 同じ機能を提供する新しいテクノロジーまたは実装に置き換える予定です。

Red Hat では、ラベル付けされていない拡張の実稼働環境での使用はサポートしていません。

9. **Generate your application** を選択して選択内容を確認し、生成されたプロジェクトを含むアーカイブのダウンロードリンクのオーバーレイ画面を表示します。オーバーレイ画面には、アプリケーションのコンパイルおよび起動に使用できるカスタムコマンドも表示されます。
10. **Download the ZIP** を選択して、生成されたプロジェクトファイルを含むアーカイブをマシンに保存します。
11. アーカイブの内容を展開します。
12. 展開したプロジェクトファイルが含まれるディレクトリーに移動します。

```
cd <directory_name>
```

13. 開発モードでアプリケーションをコンパイルして起動します。

```
./mvnw compile quarkus:dev
```

## 6.4. QUARKUS CLI を使用した RED HAT ビルドの QUARKUS MAVEN プロジェクトの作成

Quarkus コマンドラインインターフェイス (CLI) を使用して、Quarkus OptaPlanner プロジェクトを作成できます。

### 前提条件

- Quarkus CLI をインストールしている。詳細は、[Building Quarkus Apps with Quarkus Command Line Interface](#) を参照してください。

### 手順

1. Quarkus アプリケーションを作成します。

```
quarkus create app -P io.quarkus:quarkus-bom:2.2.3.Final-redhat-00013
```

2. 利用可能な拡張機能を表示するには、以下のコマンドを入力します。

```
quarkus ext -i
```

このコマンドは、以下の拡張機能を返します。

```
optaplanner-quarkus
optaplanner-quarkus-benchmark
optaplanner-quarkus-jackson
optaplanner-quarkus-jsonb
```

3. 以下のコマンドを入力して、エクステンションをプロジェクトの **pom.xml** ファイルに追加します。

```
quarkus ext add resteasy-jackson
quarkus ext add optaplanner-quarkus
quarkus ext add optaplanner-quarkus-jackson
```

4. テキストエディターで **pom.xml** ファイルを開きます。ファイルの内容は以下の例のようになります。

```
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.acme</groupId>
  <artifactId>code-with-quarkus-optaplanner</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <properties>
    <compiler-plugin.version>3.8.1</compiler-plugin.version>
    <maven.compiler.parameters>true</maven.compiler.parameters>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
    <quarkus.platform.group-id>io.quarkus</quarkus.platform.group-id>
    <quarkus.platform.version>2.2.3.Final-redhat-00013</quarkus.platform.version>
    <surefire-plugin.version>3.0.0-M5</surefire-plugin.version>
  </properties>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>${quarkus.platform.artifact-id}</artifactId>
        <version>${quarkus.platform.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
      <dependency>
        <groupId>io.quarkus.platform</groupId>
        <artifactId>optaplanner-quarkus</artifactId>
        <version>2.2.2.Final</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-arc</artifactId>
    </dependency>
    <dependency>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-resteasy</artifactId>
    </dependency>
  </dependencies>
```

```

    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-quarkus</artifactId>
  </dependency>
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-quarkus-jackson</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-jackson</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-junit5</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>${quarkus.platform.group-id}</groupId>
      <artifactId>quarkus-maven-plugin</artifactId>
      <version>${quarkus.platform.version}</version>
      <extensions>true</extensions>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
            <goal>generate-code</goal>
            <goal>generate-code-tests</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${compiler-plugin.version}</version>
      <configuration>
        <parameters>${maven.compiler.parameters}</parameters>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <configuration>
        <systemPropertyVariables>
<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
        <maven.home>${maven.home}</maven.home>
      </systemPropertyVariables>
    </configuration>
  </plugins>
</build>

```



```
</plugin>
</plugins>
</build>
<profiles>
<profile>
  <id>native</id>
  <activation>
    <property>
      <name>native</name>
    </property>
  </activation>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-failsafe-plugin</artifactId>
        <version>${surefire-plugin.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>integration-test</goal>
              <goal>verify</goal>
            </goals>
            <configuration>
              <systemPropertyVariables>
                <native.image.path>${project.build.directory}/${project.build.finalName}-
runner</native.image.path>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
                <maven.home>${maven.home}</maven.home>
              </systemPropertyVariables>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <properties>
    <quarkus.package.type>native</quarkus.package.type>
  </properties>
</profile>
</profiles>
</project>
```

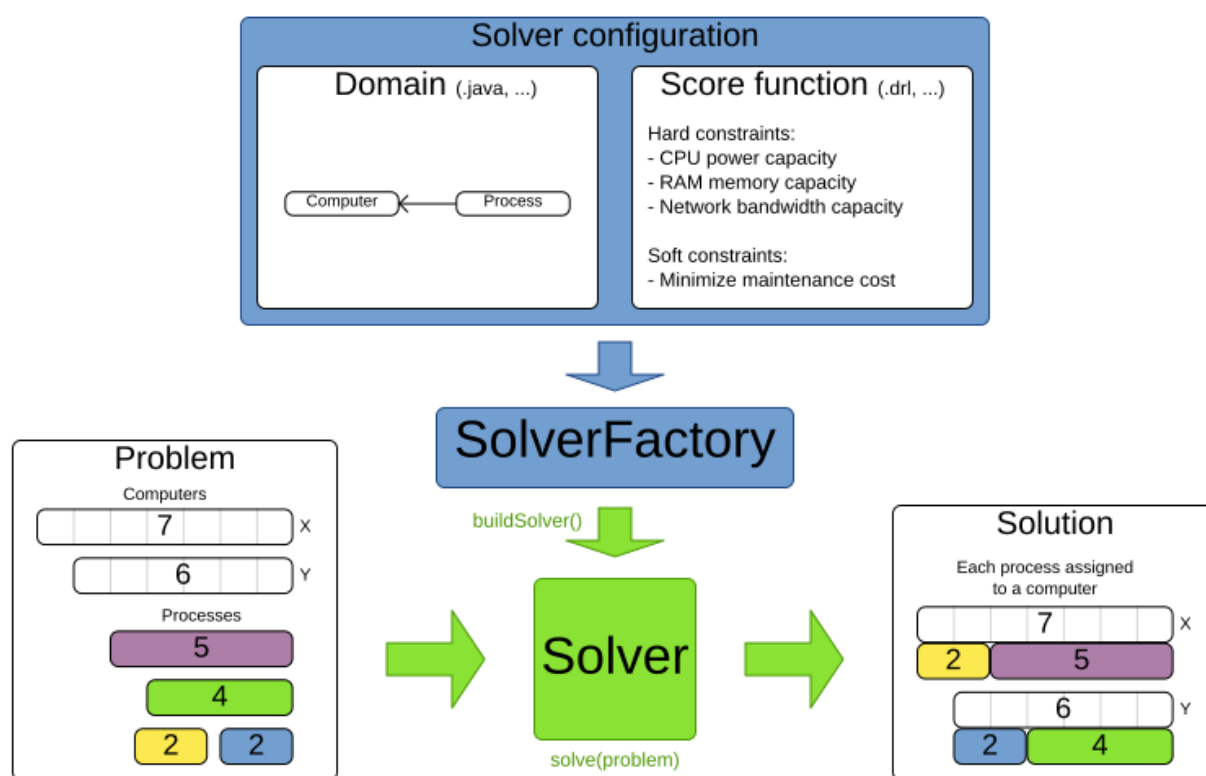
## パート III. RED HAT ビルドの OPTAPLANNER のソルバー

OptaPlanner を使用して計画の問題を解決するには、次の手順を実行します。

1. **@PlanningSolution** アノテーションでアノテーションが付けられたクラスとして **プランニングの問題をモデル化**します (例: **NQueens** クラス)。
2. **Solver** を設定します (**NQueens** インスタンスの First Fit や Tabu Search Solver など)。
3. データレイヤーから**問題データセットを読み込み**ます (例: Queens インスタンス)。これが**プランニング問題**です。
4. 見つかった最善解を返す **Solver.solve (problem)** で **解決**します。

### Input/Output overview

Use 1 SolverFactory per application and 1 Solver per dataset.



## 第7章 OPTAPLANNER ソルバーの REDHAT ビルドの設定

以下の方法を使用して、OptaPlanner のソルバーを設定できます。

- XML ファイルを使用します。
- **SolverConfig** API を使用します。
- ドメインモデルにクラスアノテーションと JavaBean プロパティアノテーションを追加します。
- OptaPlanner がドメインにアクセスするために使用するメソッドを制御します。
- カスタムプロパティを定義します。

### 7.1. XML ファイルを使用した OPTAPLANNER のソルバーの設定

XML ファイルを使用して Solver を設定できます。Maven ディレクトリー構造に従う一般的なプロジェクトでは、**SolverFactory** を使用して **Solver** インスタンスをビルドした後、**solverConfigXML** ファイルは `$PROJECT_DIR / src / main / resources / org / optaplanner / examples / <PROJECT> / solver` ディレクトリーにあります。ここで、**<PROJECT>** は OptaPlanner プロジェクトの名前です。または、**SolverFactory.createFromXmlFile()** でファイルから **SolverFactory** を作成することもできます。ただし、移植性の理由から、クラスパスのリソースが推奨されます。

**Solver** と **SolverFactory** の両方に、**Solution\_** と呼ばれるジェネリック型があります。これは、計画の問題と解決策を表すクラスです。

OptaPlanner を使用すると、設定を変更することで、最適化アルゴリズムを比較的簡単に切り替えることができます。

#### 手順

1. **SolverFactory** で **Solver** インスタンスをビルドします。
2. Solver 設定の XML ファイルを設定します。
  - a. モデルを定義します。
  - b. スコア機能を定義します。
  - c. 必要に応じて、最適化アルゴリズムを設定します。  
以下の例は、NQueens 問題に対するソルバー XML ファイルです。

```
<?xml version="1.0" encoding="UTF-8"?>
<solver xmlns="https://www.optaplanner.org/xsd/solver"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
  https://www.optaplanner.org/xsd/solver/solver.xsd">
  <!-- Define the model -->
  <solutionClass>org.optaplanner.examples.nqueens.domain.NQueens</solutionClass>
  <entityClass>org.optaplanner.examples.nqueens.domain.Queen</entityClass>

  <!-- Define the score function -->
  <scoreDirectorFactory>

  <scoreDrl>org/optaplanner/examples/nqueens/solver/nQueensConstraints.drl</scoreDrl>
```

```

</scoreDirectorFactory>

<!-- Configure the optimization algorithms (optional) -->
<termination>
...
</termination>
<constructionHeuristic>
...
</constructionHeuristic>
<localSearch>
...
</localSearch>
</solver>

```

### 注記

一部の環境では、OSGi や JBoss モジュールなどの一部の環境では、JAR ファイルのソルバー設定、スコア DRL、ドメインクラスなどのクラスパスリソースが **optaplanner-core** JAR ファイルのデフォルトの **ClassLoader** で利用できない場合もあります。このような場合は、クラスの **ClassLoader** をパラメーターとして提供します。

```

SolverFactory<NQueens> solverFactory =
SolverFactory.createFromXmlResource(
    ".../nqueensSolverConfig.xml", getClass().getClassLoader());

```

3. **ClassLoader.getResource()** で定義されているクラスパスリソースとして提供されるソルバー設定 XML ファイルを使用して **SolverFactory** を設定します。

```

SolverFactory<NQueens> solverFactory = SolverFactory.createFromXmlResource(
    "org/optaplanner/examples/nqueens/solver/nqueensSolverConfig.xml");
Solver<NQueens> solver = solverFactory.buildSolver();

```

## 7.2. JAVA API を使用した OPTAPLANNER のソルバーの設定

**SolverConfig** API を使用して Solver を設定できます。これは特に、ランタイム時に値を動的に変更する場合に便利です。以下の例では、NQueens プロジェクトで **Solver** を構築する前のシステムプロパティーに基づいて実行時間を変更します。

```

SolverConfig solverConfig = SolverConfig.createFromXmlResource(
    "org/optaplanner/examples/nqueens/solver/nqueensSolverConfig.xml");
solverConfig.withTerminationConfig(new TerminationConfig()
    .withMinutesSpentLimit(userInput));

SolverFactory<NQueens> solverFactory = SolverFactory.create(solverConfig);
Solver<NQueens> solver = solverFactory.buildSolver();

```

ソルバー設定 XML ファイルのすべての要素は、パッケージ namespace **org.optaplanner.core.config** の **Config** クラスまたは **プロパティー** として使用できます。これらの **Config** クラスは XML 形式の Java 表現です。これらのコンポーネントには、パッケージ namespace **org.optaplanner.core.impl** のランタイムコンポーネントをビルドし、それらを効率的な **Solver** に組み込むことができます。

## 注記

各ユーザー要求に **SolverFactory** を動的に設定するには、初期化中にテンプレート **SolverConfig** をビルドし、各ユーザー要求のコピーコンストラクターでコピーします。以下の例は、NQueens の問題でこれを実行する方法を示しています。

```
private SolverConfig template;

public void init() {
    template = SolverConfig.createFromXmlResource(
        "org/optaplanner/examples/nqueens/solver/nqueensSolverConfig.xml");
    template.setTerminationConfig(new TerminationConfig());
}

// Called concurrently from different threads
public void userRequest(..., long userInput) {
    SolverConfig solverConfig = new SolverConfig(template); // Copy it
    solverConfig.getTerminationConfig().setMinutesSpentLimit(userInput);
    SolverFactory<NQueens> solverFactory = SolverFactory.create(solverConfig);
    Solver<NQueens> solver = solverFactory.buildSolver();
    ...
}
```

## 7.3. OPTAPLANNER アノテーション

ドメインモデルのクラスは、プランニング変数などのプランニングエンティティを指定する必要があります。以下の方法のいずれかを使用して、OptaPlanner プロジェクトにアノテーションを追加します。

- ドメインモデルにクラスアノテーションと JavaBean プロパティアノテーションを追加します。プロパティアノテーションは setter メソッドではなく getter メソッドに配置する必要があります。アノテーションが付けられた getter メソッドの公開は必要ありません。これは推奨される方法です。
- ドメインモデルにクラスアノテーションとフィールドアノテーションを追加します。アノテーションが付けられたフィールドはパブリックである必要はありません。

## 7.4. OPTAPLANNER ドメインアクセスの指定

デフォルトでは、OptaPlanner はリフレクションを使用してドメインにアクセスします。リフレクションは信頼性がありますが、直接アクセスに比べると遅くなります。または、Gizmo を使用してドメインにアクセスするように OptaPlanner を設定できます。これにより、リフレクションなしにドメインのフィールドとメソッドに直接アクセスするバイトコードが生成されます。ただし、この手法には以下の制限があります。

- プランニングアノテーションは、パブリックフィールドおよびパブリック getter でのみ指定できます。
- `io.quarkus.gizmo:gizmo` はクラスパス上にある必要があります。

## 注記

Gizmo がデフォルトのドメインアクセスタイプであるため、Quarkus で OptaPlanner を使用する場合、これらの制限は適用されません。

## 手順

Quarkus の外部にある Gizmo を使用するには、ソルバー設定で **domainAccessType** を設定します。

```

<solver>
  <domainAccessType>GIZMO</domainAccessType>
</solver>

```

## 7.5. カスタムプロパティの設定

OptaPlanner プロジェクトでは、クラスをインスタンス化し、カスタムプロパティに明示的に言及するドキュメントを持つソルバー設定要素にカスタムプロパティを追加できます。

### 前提条件

- ソルバーがあること。

### 手順

1. カスタムプロパティを追加します。  
たとえば、**Easy ScoreCalculator** にキャッシュされる大きな計算があり、1つのベンチマークでキャッシュサイズを増やす場合は、**myCacheSize** プロパティを追加します。

```

<scoreDirectorFactory>
  <easyScoreCalculatorClass>...MyEasyScoreCalculator</easyScoreCalculatorClass>
  <easyScoreCalculatorCustomProperties>
    <property name="myCacheSize" value="1000"/><!-- Override value -->
  </easyScoreCalculatorCustomProperties>
</scoreDirectorFactory>

```

2. カスタムプロパティごとにパブリックセッターを追加します。これは、**ソルバーの** ビルド時に呼び出されます。

```

public class MyEasyScoreCalculator extends EasyScoreCalculator<MySolution,
SimpleScore> {

    private int myCacheSize = 500; // Default value

    @SuppressWarnings("unused")
    public void setMyCacheSize(int myCacheSize) {
        this.myCacheSize = myCacheSize;
    }

    ...
}

```

**boolean**、**int**、**double**、**BigDecimal**、**String**、**enums** など、ほとんどの値型がサポートされています。

## 第8章 OPTAPLANNER ソルバー

ソルバーは、計画の問題に対する最適で最適なソリューションを見つけます。ソルバーで解決できる計画問題インスタンスは度に1つずつです。ソルバーは、**SolverFactory** メソッドを使用して構築されます。

```
public interface Solver<Solution_> {

    Solution_ solve(Solution_ problem);

    ...

}
```

スレッドセーフであることが **javadoc** に具体的に記載されているメソッドを除いて、ソルバーには単一のスレッドからアクセスする必要があります。**solve()** メソッドは、現在のスレッドを占有します。スレッドを占有すると、REST サービスの HTTP タイムアウトが発生する可能性があり、複数のデータセットを並行して解決するために追加のコードが必要になります。このような問題を回避するには、代わりに **SolverManager** を使用してください。

### 8.1. 問題の解決

ソルバーを使用して、計画の問題を解決します。

#### 前提条件

- ソルバー設定で構築された **Solver**
- 計画問題インスタンスを表す **@PlanningSolution** アノテーション

#### 手順

計画問題を **solve()** メソッドの引数として提供します。ソルバーは、見つかった最良の解を返します。

次の例は、NQueens の問題を解決します。

```
NQueens problem = ...;
NQueens bestSolution = solver.solve(problem);
```

この例では、**solve()** メソッドは、すべての **Queen** が **Row** に割り当てられた **NQueens** インスタンスを返します。







#### 注記

**solve (Solution)** メソッドに指定されたソリューションインスタンスは、部分的または完全に初期化できます。これは、繰り返して計画する場合によくあります。



図8.1 8ms のフォークイーンズパズルのベストソリューション (最適なソリューションでもあります)

	A	B	C	D
0				
1				
2				
3				

**solve (Solution)** メソッドは、問題のサイズとソルバーの設定によっては時間がかかる場合があります。**Solver** は、可能な解決策の検索スペースをインテリジェントに処理し、解決中に遭遇した最良の解決策を記憶します。問題の大きさ、**Solver** が持っているどのくらいの時間、ソルバーの設定、等々の数ある要因により、**最善の** 解決策は、**最適な** 解決策ではない可能性もあります、



### 注記

メソッド **solve (Solution)** に与えられたソリューションインスタンスは **Solver** によって変更されますが、それを最良のソリューションと間違えないでください。

**solve (Solution)** または **getBestSolution()** メソッドによって返されたソリューションのインスタンスは、**solve (Solution)** に渡された計画インスタンスのクローンである可能性が高いです。この場合、つまりは別のインスタンスであることを意味しています。

## 8.2. ソルバー環境モード

ソルバー環境モードを使用すると、実装の一般的なバグを検出できます。ロギングレベルには影響しません。

ソルバーには単一のランダムインスタンスがあります。一部のソルバー設定は、他の設定よりもランダムインスタンスを多く使用します。たとえば、シミュレーテッドアニーリングアルゴリズムは乱数に大きく依存しますが、**Tabu Search** はスコアの同点を解決するために乱数にのみ依存します。環境モードは、そのランダムインスタンスのシードに影響を与えます。

ソルバー設定 XML ファイルで環境モードを設定できます。次の例では、**FAST\_ASSERT** モードを設定します。

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <environmentMode>FAST_ASSERT</environmentMode>
  ...
</solver>
```

次のリストは、ソルバー設定ファイルで使える環境モードについて説明しています。

- **FULL\_ASSERT** モードは、すべてのアサーションをオンにします。たとえば、増分スコアの計算が移動ごとに破損していないというアサーション、Move 実装のバグ、制約、エンジン自体などでフェイルファストします。このモードは再現可能です。また、非アサートモードよりも



頻繁に `calculateScore()` メソッドを呼び出すため、煩わしいものです。**FULL\_ASSERT** モードは、増分スコア計算に依存しないため、非常に低速です。

- **NON\_INTRUSIVE\_FULL\_ASSERT** モードは、Move 実装のバグ、制約、エンジン自体などでフェイルファストするためにいくつかのアサーションをオンにします。このモードは再現可能です。非アサートモードよりも頻繁に `calculateScore()` メソッドを呼び出さないため、邪魔になりません。**NON\_INTRUSIVE\_FULL\_ASSERT** モードは、増分スコア計算に依存しないため、非常に低速です。
- **FAST\_ASSERT** モードは、undoMove のスコアが Move の前と同じであるというアサーションなど、ほとんどのアサーションをオンにして、Move 実装のバグ、制約、エンジン自体などをフェイルファストします。このモードは再現可能です。また、非アサートモードよりも頻繁に `calculateScore()` メソッドを呼び出すため、煩わしいものです。**FAST\_ASSERT** モードは遅いです。**FAST\_ASSERT** モードをオンにして、計画の問題を短時間実行するテストケースを作成します。
- **REPRODUCIBLE** モードは、開発中に推奨されるため、デフォルトのモードです。このモードでは、同じ OptaPlanner バージョンで 2 回実行すると、同じコードが同じ順序で実行されます。次の注意事項が当てはまる場合を除いて、これら 2 つの実行はすべてのステップで同じ結果になります。これにより、バグを一貫して再現できます。また、スコア制約のパフォーマンスの最適化など、特定のリファクタリングを実行全体で公平にベンチマークすることもできます。



#### 注記

**REPRODUCIBLE** モードを使用しているにもかかわらず、次の理由により、アプリケーションが完全に再現できない場合があります。

- 特にソリューションの実装において、計画エンティティまたは計画値のコレクションに対して、JVM 実行間で順序が一貫していないが、通常の問題の事実ではない **HashSet** または別の **コレクション** を使用する。**LinkedHashSet** に置き換えます。
  - 時間勾配に依存するアルゴリズム、特にシミュレーテッドアニーリングアルゴリズムを、終了に費やした時間と組み合わせます。割り当てられた CPU 時間大きな違いがあると、時間勾配値に影響を与えます。シミュレーテッドアニーリングアルゴリズムをレイトアクセプタンスアルゴリズムに置き換えるか、終了に費やした時間をステップカウント終了に置き換えます。
- **REPRODUCIBLE** モードは、**NON\_REPRODUCIBLE** モードよりもわずかに遅くなる可能性があります。実稼働環境で再現性の恩恵を受けることができる場合は、実稼働でこのモードを使用してください。実際には、**REPRODUCIBLE** モードでは、シードが指定されていない場合、デフォルトの固定ランダムシードが使用され、ワークスティーリングなどの特定の同時実行の最適化も無効になります。
  - **NON\_REPRODUCIBLE** モードは、**REPRODUCIBLE** モードよりもわずかに高速です。デバッグやバグ修正が困難になるため、開発中の使用は避けてください。実稼働環境で再現性が重要でない場合は、実稼働で **NON\_REPRODUCIBLE** モードを使用してください。実際には、シードが指定されていない場合、このモードは固定ランダムシードを使用しません。

### 8.3. OPTAPLANNER ソルバーのログレベルの変更

OptaPlanner ソルバーのログレベルを変更して、ソルバーアクティビティを確認できます。次のリストは、さまざまなログレベルについて説明しています。

- **error:RuntimeException** として呼び出し元のコードに throw されるエラーを除いて、エラーをログに記録します。  
エラーが発生した場合、OptaPlanner は通常は短時間で失敗します。呼び出し元のコードに詳細なメッセージを含む **RuntimeException** の サブクラスを出力します。ログメッセージの重複を避けるために、エラーとしてログに記録されません。**呼び出し元のコードがその RuntimeException を明示的にキャッチして排除しない限り、スレッドのデフォルトのExceptionHandler はとにかくそれをエラーとしてログに記録します。その間、コードはさらに害を及ぼしたり、エラーを難読化したりすることで中断されます。**
- **警告:** 疑わしい状況をログに記録します
- **info:** すべてのフェーズとソルバー自体をログに記録します
- **デバッグ:** すべてのフェーズのすべてのステップをログに記録します
- **トレース:** すべてのフェーズのすべてのステップのすべての動きをログに記録します

## 注記

**トレース** ログを指定すると、パフォーマンスが大幅に低下します。ただし、**トレース** ロギングは、ボトルネックを発見するための開発中に非常に重要です。

**デバッグ** ログでさえ、レイトアクセプタンスやシミュレーテッドアニーリングなどの高速ステッピングアルゴリズムではパフォーマンスが大幅に低下する可能性があります。タブーサーチなどの低速ステッピングアルゴリズムでは低下しません。

`trace`` と **デバッグ** ロギングの両方が、ほとんどのアペンダーでのマルチスレッド解決で輻輳を引き起こします。

Eclipse では、コンソールへの **デバッグ** ログにより、スコア計算速度が1秒あたり10000 を超える輻輳が発生する傾向があります。IntelliJ も Maven コマンドラインもこの問題に悩まされていません。

## 手順

ロギングレベルを **デバッグ** ロギングに設定して、フェーズがいつ終了し、どのくらいの速さでステップが実行されるかを確認します。

次の例は、デバッグログからの出力を示しています。

```
INFO Solving started: time spent (3), best score (-4init/0), random (JDK with seed 0).
DEBUG CH step (0), time spent (5), score (-3init/0), selected move count (1), picked move
(Queen-2 {null -> Row-0}).
DEBUG CH step (1), time spent (7), score (-2init/0), selected move count (3), picked move
(Queen-1 {null -> Row-2}).
DEBUG CH step (2), time spent (10), score (-1init/0), selected move count (4), picked move
(Queen-3 {null -> Row-3}).
DEBUG CH step (3), time spent (12), score (-1), selected move count (4), picked move (Queen-0
{null -> Row-1}).
INFO Construction Heuristic phase (0) ended: time spent (12), best score (-1), score calculation
speed (9000/sec), step total (4).
DEBUG LS step (0), time spent (19), score (-1), best score (-1), accepted/selected move count
(12/12), picked move (Queen-1 {Row-2 -> Row-3}).
DEBUG LS step (1), time spent (24), score (0), new best score (0), accepted/selected move count
(9/12), picked move (Queen-3 {Row-3 -> Row-2}).
INFO Local Search phase (1) ended: time spent (24), best score (0), score calculation speed
```

(4000/sec), step total (2).

INFO Solving ended: time spent (24), best score (0), score calculation speed (7000/sec), phase total (2), environment mode (REPRODUCIBLE).

費やされた時間の値はすべてミリ秒単位です。

すべてが [SLF4J](#) に記録されます。これは、すべてのログメッセージを Logback、Apache Commons Logging、Log4j、または `java.util.logging` に委任する単純なログファサードです。選択したロギングフレームワークのロギングアダプターに依存関係を追加します。

## 8.4. LOGBACK を使用して OPTAPLANNER ソルバーアクティビティをログに記録する

Logback は、OptaPlanner で使用するために推奨されるロギングフレームワークです。Logback を使用して、OptaPlanner ソルバーアクティビティをログに記録します。

### 前提条件

- OptaPlanner プロジェクトがあります。

### 手順

1. 次の Maven 依存関係を OptaPlanner プロジェクトの **pom.xml** ファイルに追加します。



#### 注記

ブリッジの依存関係を追加する必要はありません。

```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.x</version>
</dependency>
```

2. 次の例に示すように、**logback.xml** ファイルの **org.optaplanner** パッケージのログレベルを設定します。ここで、**<LEVEL>** は「[Logback を使用して OptaPlanner ソルバーアクティビティをログに記録する](#)」にリストされているログレベルです。。

```
<configuration>

  <logger name="org.optaplanner" level="<LEVEL>"/>

  ...

</configuration>
```

3. オプション: 複数の **ソルバー** インスタンスが同時に実行されている可能性があるマルチテナントアプリケーションがある場合は、各インスタンスのログを別々のファイルに分割します。
  - a. **solve()** 呼び出しを **マップされた診断コンテキスト** (MDC) で囲みます。

```
MDC.put("tenant.name",tenantName);
MySolution bestSolution = solver.solve(problem);
MDC.remove("tenant.name");
```

- b. **\${tenant.name}** ごとに異なるファイルを使用するようにロガーを設定します。たとえば、**logback.xml** ファイルで **SiftingAppender** を使用します。

```
<appender name="fileAppender" class="ch.qos.logback.classic.sift.SiftingAppender">
  <discriminator>
    <key>tenant.name</key>
    <defaultValue>unknown</defaultValue>
  </discriminator>
  <sift>
    <appender name="fileAppender.${tenant.name}" class="...FileAppender">
      <file>local/log/optaplanner-${tenant.name}.log</file>
    ...
  </appender>
</sift>
</appender>
```



#### 注記

複数のソルバーまたは1つのマルチスレッドソルバーを実行する場合、コンソールを含むほとんどのアペンダーは、**デバッグ** および **トレース** ログで輻輳を引き起こします。この問題を回避するには、非同期アペンダーに切り替えるか、**デバッグ** ログをオフにします。

4. OptaPlanner が新しいレベルを認識しない場合は、一時的にシステムプロパティ - **Dlogback.LEVEL=true** を追加してトラブルシューティングします。

## 8.5. LOG4J を使用して OPTAPLANNER ソルバーアクティビティをログに記録する

すでに Log4J を使用していて、より高速な後継である Logback に切り替えたくない場合は、Log4J 用に OptaPlanner プロジェクトを設定できます。

### 前提条件

- OptaPlanner プロジェクトがあります
- Log4J ロギングフレームワークを使用しています

### 手順

1. ブリッジの依存関係をプロジェクトの **pom.xml** ファイルに追加します。

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.x</version>
</dependency>
```

2. 次の例に示すように、**log4j.xml** ファイルのパッケージ **org.optaplanner** でログレベルを設定します。ここで、**<LEVEL>** は「[Logback を使用して OptaPlanner ソルバーアクティビティをログに記録する](#)」にリストされているログレベルです。

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

  <category name="org.optaplanner">
    <priority value="<LEVEL>" />
  </category>

  ...

</log4j:configuration>
```

3. オプション: 複数の **ソルバー** インスタンスが同時に実行されている可能性があるマルチテナントアプリケーションがある場合は、各インスタンスのログを別々のファイルに分割します。
- a. **solve()** 呼び出しを **マップされた診断コンテキスト (MDC)** で囲みます。

```
MDC.put("tenant.name",tenantName);
MySolution bestSolution = solver.solve(problem);
MDC.remove("tenant.name");
```

- b. **\${tenant.name}** ごとに異なるファイルを使用するようにロガーを設定します。たとえば、**logback.xml** ファイルで **SiftingAppender** を使用します。

```
<appender name="fileAppender" class="ch.qos.logback.classic.sift.SiftingAppender">
  <discriminator>
    <key>tenant.name</key>
    <defaultValue>unknown</defaultValue>
  </discriminator>
  <sift>
    <appender name="fileAppender.${tenant.name}" class="...FileAppender">
      <file>local/log/optaplanner-${tenant.name}.log</file>
    ...
  </appender>
</sift>
</appender>
```



### 注記

複数のソルバーまたは1つのマルチスレッドソルバーを実行する場合、コンソールを含むほとんどのアペンダーは、**デバッグ** および **トレース** ログで輻輳を引き起こします。この問題を回避するには、非同期アペンダーに切り替えるか、**デバッグ** ログをオフにします。

## 8.6. ソルバーの監視

OptaPlanner は、Java アプリケーション用のメトリック計測ライブラリーである **Micrometer** を介してメトリックを公開します。一般的な監視システムで Micrometer を使用して、OptaPlanner ソルバーを監視できます。

### 8.6.1. Micrometer 用の Quarkus OptaPlanner アプリケーションの設定

OptaPlanner Quarkus アプリケーションを Micrometer および指定された監視システムを使用するように設定するには、Micrometer 依存関係を **pom.xml** ファイルに追加します。

### 前提条件

- Quarkus OptaPlanner アプリケーションがあります。

### 手順

1. 次の依存関係をアプリケーションの **pom.xml** ファイルに追加します。ここで **<MONITORING\_SYSTEM>** は Micrometer と Quarkus でサポートされている監視システムです。



#### 注記

Prometheus は現在、Quarkus でサポートされている唯一の監視システムです。

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-micrometer-registry-<MONITORING_SYSTEM></artifactId>
</dependency>
```

2. アプリケーションを開発モードで実行するには、次のコマンドを入力します。

```
mvn compile quarkus:dev
```

3. アプリケーションのメトリックを表示するには、ブラウザに次の URL を入力します。

```
http://localhost:8080/q/metrics
```

## 8.6.2. Micrometer 用の Spring Boot OptaPlanner アプリケーションの設定

Micrometer と指定された監視システムを使用するように Spring Boot OptaPlanner アプリケーションを設定するには、**Pom.xml** ファイルに Micrometer 依存関係を追加します。

### 前提条件

- Spring Boot OptaPlanner アプリケーションがあります。

### 手順

1. 次の依存関係をアプリケーションの **pom.xml** ファイルに追加します。ここで **<MONITORING\_SYSTEM>** は Micrometer と Spring Boot でサポートされている監視システムです。

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
```

```
<groupId>io.micrometer</groupId>
<artifactId>micrometer-registry-<MONITORING_SYSTEM></artifactId>
</dependency>
```

2. アプリケーションの **application.properties** ファイルに設定情報を追加します。詳細は、[Micrometer](#) Web サイトを参照してください。
3. アプリケーションを実行するには、以下のコマンドを入力します。

```
mvn spring-boot:run
```

4. アプリケーションのメトリックを表示するには、ブラウザーに次の URL を入力します。  
<http://localhost:8080/actuator/metrics>



#### 注記

次の URL を Prometheus スクレイパーパスとして使用します:  
<http://localhost:8080/actuator/prometheus>

### 8.6.3. Micrometer 用のプレーンな Java OptaPlanner アプリケーションの設定

Micrometer を使用するようにプレーンな Java OptaPlanner アプリケーションを設定するには、Micrometer の依存関係と、選択した監視システムの設定情報をプロジェクトの **POM.XML** ファイルに追加する必要があります。

#### 前提条件

- プレーンな Java OptaPlanner アプリケーションがあります。

#### 手順

1. 次の依存関係をアプリケーションの **pom.xml** ファイルに追加します。ここで、**<MONITORING\_SYSTEM>** は Micrometer で設定された監視システムであり、**<VERSION>** は使用している Micrometer のバージョンです。

```
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-<MONITORING_SYSTEM></artifactId>
  <version><VERSION></version>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-core</artifactId>
  <version>`<VERSION>`</version>
</dependency>
```

2. プロジェクトの **pom.xml** ファイルの先頭に、監視システムの Micrometer 設定情報を追加します。詳細は、[Micrometer](#) Web サイトを参照してください。
3. 設定情報の下に次の行を追加します。ここで、**<MONITORING\_SYSTEM>** は追加した監視システムです。

```
Metrics.addRegistry(<MONITORING_SYSTEM>);
```

次の例は、Prometheus 監視システムを追加する方法を示しています。

```
PrometheusMeterRegistry prometheusRegistry = new
PrometheusMeterRegistry(PrometheusConfig.DEFAULT);
try {
    HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
    server.createContext("/prometheus", httpExchange -> {
        String response = prometheusRegistry.scrape();
        httpExchange.sendResponseHeaders(200, response.getBytes().length);
        try (OutputStream os = httpExchange.getResponseBody()) {
            os.write(response.getBytes());
        }
    });
    new Thread(server::start).start();
} catch (IOException e) {
    throw new RuntimeException(e);
}
Metrics.addRegistry(prometheusRegistry);
```

4. 監視システムを開いて、OptaPlanner プロジェクトのメトリックを表示します。次のメトリックが公開されます。



#### 注記

メトリックの名前と形式は、レジストリーによって異なります。

- **optaplanner.solver.errors.total**: 測定開始以降に解決中に発生したエラーの総数。
- **optaplanner.solver.solve-length.active-count**: 現在解いているソルバーの数。
- **optaplanner.solver.solve-length.seconds-max**: 現在アクティブなソルバーの実行時間が最も長い実行時間。
- **optaplanner.solver.solve-length.seconds-duration-sum**: アクティブな各ソルバーの解決時間の合計。たとえば、アクティブなソルバーが2つあり、一方が3分間実行され、もう一方が1分間実行されている場合、合計計算時間は4分です。

## 8.7. 乱数ジェネレーターの設定

多くのヒューリスティックおよびメタヒューリスティックは、移動の選択、スコアの結びつきの解決、確率ベースの移動の受け入れなどを疑似乱数ジェネレーターに依存しています。解決中に、同じランダムインスタンスが再利用され、再現性、パフォーマンス、およびランダム値の均一な分布が向上します。

ランダムシードは、疑似乱数ジェネレータを初期化するために使用される番号です。

### 手順

1. オプション: ランダムインスタンスのランダムシードを変更するには、**randomSeed** を指定します。

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
```



```

https://www.optaplanner.org/xsd/solver/solver.xsd">
  <randomSeed>0</randomSeed>
  ...
</solver>

```

2. オプション: 疑似乱数ジェネレーターの実装を変更するには、以下のソルバー設定ファイルにリストされている **randomType** プロパティの値を指定します。ここで、**<RANDOM\_NUMBER\_GENERATOR>** は疑似乱数ジェネレーターです。

```

<solver xmlns="https://www.optaplanner.org/xsd/solver"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <randomType><RANDOM_NUMBER_GENERATOR></randomType>
  ...
</solver>

```

次の疑似乱数ジェネレータがサポートされています。

- **JDK** (デフォルト): 標準の乱数ジェネレーターの実装 (**java.util.Random**)
- **MERSENNE\_TWISTER**: [コモンズ数学](#) によって乱数ジェネレータの実装
- **WELL512A, WELL1024A, WELL19937A, WELL19937C, WELL44497A と WELL44497B**: [Commons Math](#) による乱数ジェネレータの実装

ほとんどのユースケースでは、**randomType** プロパティの値は、複数のデータセットでの最良のソリューションの平均品質に大きな影響を与えません。

## 第9章 OPTAPLANNER SOLVERMANAGER

**SolverManager** は、REST およびその他のエンタープライズサービスの計画問題の解決を簡素化するための1つ以上の **ソルバー** インスタンスのファサードです。

**Solver.solve (...)** メソッドとは異なり、**SolverManager** は、次の特性があります。

- **SolverManager.solve (...)** はすぐに戻ります。呼び出し元のスレッドをブロックせずに、非同期解決の問題をスケジュールします。これにより、HTTP およびその他のテクノロジーのタイムアウトの問題が回避されます。
- **SolverManager.solve (...)** は、同じドメインの複数の計画問題を並行して解決します。

内部的には、**SolverManager** は、**Solver.solve (...)** を呼び出すソルバースレッドのスレッドプールと、最適なソリューション変更イベントを処理するコンシューマースレッドのスレッドプールを管理します。

Quarkus と SpringBoot では、**SolverManager** インスタンスがコードに自動的に挿入されます。Quarkus または SpringBoot 以外のプラットフォームを使用している場合は、**create (...)** メソッドを使用して **SolverManager** インスタンスをビルドします。

```
SolverConfig solverConfig =
    SolverConfig.createFromXmlResource("../cloudBalancingSolverConfig.xml");
SolverManager<CloudBalance, UUID> solverManager = SolverManager.create(solverConfig, new
    SolverManagerConfig());
```

**SolverManager.solve (...)** メソッドに送信される各問題には、一意の問題 ID が必要です。後で **getSolverStatus (problemId)** または **terminateEarly (problemId)** を呼び出すと、その問題 ID を使用して計画の問題を区別します。問題 ID は、**Long**、**String**、**java.util.UUID** などのイミュータブルなクラスである必要があります。

**SolverManagerConfig** クラスには、並行して実行されるソルバーの数を制御する **parallelSolverCount** プロパティがあります。たとえば、**parallelSolverCount** プロパティが **4** に設定されていて、5つの問題を送信すると、4つの問題がすぐに解決を開始し、最初の問題の1つが終了すると5番目の問題が開始します。これらの問題がそれぞれ5分間解決した場合、5番目の問題は10分で終了します。デフォルトでは、**parallelSolverCount** は **AUTO** に設定されており、ソルバーの **moveThreadCount** に関係なく、CPU コアの半分に解決されます。

最適なソリューションを取得するには、終了を解決した後、通常は **SolverJob.getFinalBestSolution()** を使用します。

```
CloudBalance problem1 = ...;
UUID problemId = UUID.randomUUID();
// Returns immediately
SolverJob<CloudBalance, UUID> solverJob = solverManager.solve(problemId, problem1);
...
CloudBalance solution1;
try {
    // Returns only after solving terminates
    solution1 = solverJob.getFinalBestSolution();
} catch (InterruptedException | ExecutionException e) {
    throw ...;
}
```

ただし、ユーザーがソリューションを必要とする前にバッチの問題を解決する方法と、ユーザーがソリューションを積極的に待っている間にライブで解決する方法の両方について、より良いアプローチがあります。

現在の **SolverManager** 実装は単一のコンピューターノードで実行されますが、将来の作業は、クラウド全体にソルバーの負荷を分散することを目的としています。

## 9.1. 問題のバッチ解決

バッチ解決とは、複数のデータセットを並行して解決することです。バッチ解決は夜間処理で特に役立ちます。

- 通常、深夜には問題の変化はほとんどまたはまったくありません。一部の組織は期限を強制します。たとえば、**深夜零時までに休日のリクエストを送信する**といったものです。
- ソルバーは、結果を待つ人が誰もいないため、CPU リソースが安価であることが多いため、はるかに長く、多くの場合は数時間実行できます。
- 翌営業日に従業員が職場に到着したときにソリューションが利用できます。

### 手順

**parallelSolverCount** によって制限した上で並列バッチでの問題解決するには、各データセットのためには、以下のクラスによって作成した以下の各データについて **solve (...)** を呼び出します。

+

```
public class TimeTableService {

    private SolverManager<TimeTable, Long> solverManager;

    // Returns immediately, call it for every data set
    public void solveBatch(Long timeTableId) {
        solverManager.solve(timeTableId,
            // Called once, when solving starts
            this::findById,
            // Called once, when solving ends
            this::save);
    }

    public TimeTable findById(Long timeTableId) {...}

    public void save(TimeTable timeTable) {...}

}
```

## 9.2. 解決して進捗状況を確認する

ユーザーがソリューションを待っている間にソルバーが実行されている場合、ユーザーは結果を受け取るまでに数分または数時間待つ必要がある場合があります。すべてが順調に進んでいることをユーザーに保証するために、これまでに達成された最良の解決策と最良のスコアを表示して進捗状況を示します。

### 手順

1. 中間の最良のソリューションを処理するには、**solveAndListen(...)** を使用します。

```
public class TimeTableService {

    private SolverManager<TimeTable, Long> solverManager;

    // Returns immediately
    public void solveLive(Long timeTableId) {
        solverManager.solveAndListen(timeTableId,
            // Called once, when solving starts
            this::findById,
            // Called multiple times, for every best solution change
            this::save);
    }

    public TimeTable findById(Long timeTableId) {...}

    public void save(TimeTable timeTable) {...}

    public void stopSolving(Long timeTableId) {
        solverManager.terminateEarly(timeTableId);
    }

}
```

この実装は、データベースを使用して、データベースをポーリングする UI と通信します。より高度な実装は、最適なソリューションを UI またはメッセージングキューに直接プッシュします。

2. ユーザーが中間の最良の解決策に満足し、より良い解決策をこれ以上待ちたくない場合は、**SolverManager.terminateEarly (problemId)** を呼び出します。

## パート IV. RED HAT ビルドの OPTAPLANNER クイックスタートガイド

以下の手順に従って、従業員の勤務表サンプルに、ShiftAssignment データオブジェクトをプランニングエンティティとして定義します。

- Red Hat ビルドの Quarkus 上の Red Hat ビルドの OptaPlanner: 時間割のクイックスタートガイド
- Red Hat ビルドの Quarkus での Red Hat ビルドの OptaPlanner: ワクチン接種予約スケジューラーのクイックスタートガイド
- Spring Boot 上の Red Hat ビルドの OptaPlanner: 時間割のクイックスタートガイド
- Red Hat ビルドの OptaPlanner と Java Solver: クラウドバランシングのクイックスタートガイド

## 第10章 RED HAT ビルドの QUARKUS 上の RED HAT ビルドの OPTAPLANNER: 時間割のクイックスタートガイド

本書では、Red Hat ビルドの OptaPlanner の制約解決人工知能 (AI) を使用して Red Hat ビルドの Quarkus アプリケーションを作成するプロセスを説明します。学生および教師向けの時間割を最適化する REST アプリケーションを構築していきます。

Refresh	Solve	Score: 0hard/18soft	By room	By teacher	By student group
Timeslot	Room A	Room B	Room C		
Monday 08:30 - 09:30		Physics by M. Curie 10th grade 27	Spanish by P. Cruz 9th grade 22		
Monday 09:30 - 10:30		Physics by M. Curie 9th grade 16	Spanish by P. Cruz 10th grade 33		
Monday 10:30 - 11:30	Geography by C. Darwin 10th grade 30	Chemistry by M. Curie 9th grade 17			
Monday 13:30 - 14:30		Math by A. Turing 10th grade 26	English by L. Jones 9th grade 20		
Monday 14:30 - 15:30		Math by A. Turing 10th grade 25	English by L. Jones 9th grade 21		

サービスは、AI を使用して、以下のハードおよびソフトの **スケジュール制約** に準拠し、**Lesson** インスタンスを **Timeslot** インスタンスと **Room** インスタンスに自動的に割り当てます。

- 1 部屋に同時に割り当てることができる授業は、最大1コマです。
- 教師が同時に一度に行うことができる授業は最大1回です。
- 生徒は同時に出席できる授業は最大1コマです。
- 教師は、1つの部屋での授業を希望します。
- 教師は、連続した授業を好み、授業間に時間が空くのを嫌います。

数学的に考えると、学校の時間割は **NP 困難** の問題であります。つまり、スケーリングが困難です。総当たり攻撃で考えられる組み合わせを単純にすべて反復すると、スーパーコンピューターを使用したとしても、非自明的なデータセットを取得するのに数百年かかります。幸い、Red Hat ビルドの OptaPlanner などの AI 制約ソルバーには、妥当な時間内にほぼ最適なソリューションを提供する高度なアルゴリズムがあります。妥当な期間として考慮される内容は、問題の目的によって異なります。

### 前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスマーポータル (ログインが必要) の [ソフトウェアダウンロード](#) ページから入手できます。
- Apache Maven 3.6 以降がインストールされている。Maven は [Apache Maven Project](#) の Web サイトから入手できます。
- IntelliJ IDEA、VSCode、Eclipse、NetBeans などの IDE が利用できる。

## 10.1. MAVEN プラグインを使用した OPTAPLANNER RED HAT ビルドの QUARKUS MAVEN プロジェクトの作成

Apache Maven および Quarkus Maven プラグインを使用して、Red Hat ビルドの OptaPlanner および Quarkus アプリケーションの使用を開始できます。

### 前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスマーポータル (ログインが必要) の [ソフトウェアダウンロード](#) ページから入手できます。
- Apache Maven 3.6 以降がインストールされている。Maven は [Apache Maven Project](#) の Web サイトから入手できます。

### 手順

1. コマンドターミナルで以下のコマンドを入力し、Maven が JDK 11 を使用していること、そして Maven のバージョンが 3.6 以上であることを確認します。

```
mvn --version
```

2. 上記のコマンドで JDK 11 が返されない場合は、JDK 11 へのパスを PATH 環境変数に追加し、上記のコマンドを再度入力します。
3. Quarkus OptaPlanner クイックスタートプロジェクトを生成するには、以下のコマンドを入力します。

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:2.2.3.Final-redhat-00013:create \
  -DprojectId=com.example \
  -DprojectArtifactId=optaplanner-quickstart \
  -Dextensions="resteasy,resteasy-jackson,optaplanner-quarkus,optaplanner-quarkus-jackson" \
  -DplatformGroupId=com.redhat.quarkus.platform \
  -DplatformVersion=2.2.3.Final-redhat-00013 \
  -DnoExamples
```

このコマンドは、**./optaplanner-quickstart** ディレクトリーで以下の要素を作成します。

- Maven の構造
- **src/main/docker** の **Dockerfile** ファイルの例
- アプリケーションの設定ファイル

表10.1 mvn io.quarkus:quarkus-maven-plugin:2.2.3.Final-redhat-00013:create コマンドで使用したプロパティ

プロパティ	説明
<b>projectGroupId</b>	プロジェクトのグループ ID。
<b>projectArtifactId</b>	プロジェクトのアーティファクト ID。
<b>extensions</b>	このプロジェクトで使用する Quarkus 拡張のコンマ区切りリスト。Quarkus 拡張の全一覧については、特定のコマンドラインで <b>mvn quarkus:list-extensions</b> を入力します。
<b>noExamples</b>	テストまたはクラスを使用せずに、プロジェクト構造でプロジェクトを作成します。

**projectGroupId** および **projectArtifactId** プロパティの値を使用して、プロジェクトバージョンを生成します。デフォルトのプロジェクトバージョンは **1.0.0-SNAPSHOT** です。

4. OptaPlanner プロジェクトを表示するには、OptaPlanner Quickstarts ディレクトリーに移動します。

```
cd optaplanner-quickstart
```

5. **pom.xml** ファイルを確認します。コンテンツの例を以下に示します。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.quarkus.platform</groupId>
      <artifactId>quarkus-bom</artifactId>
      <version>2.2.3.Final-redhat-00013</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>io.quarkus.platform</groupId>
      <artifactId>quarkus-optaplanner-bom</artifactId>
      <version>2.2.3.Final-redhat-00013</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-jackson</artifactId>
  </dependency>
  <dependency>
    <groupId>org.optaplanner</groupId>
```



```

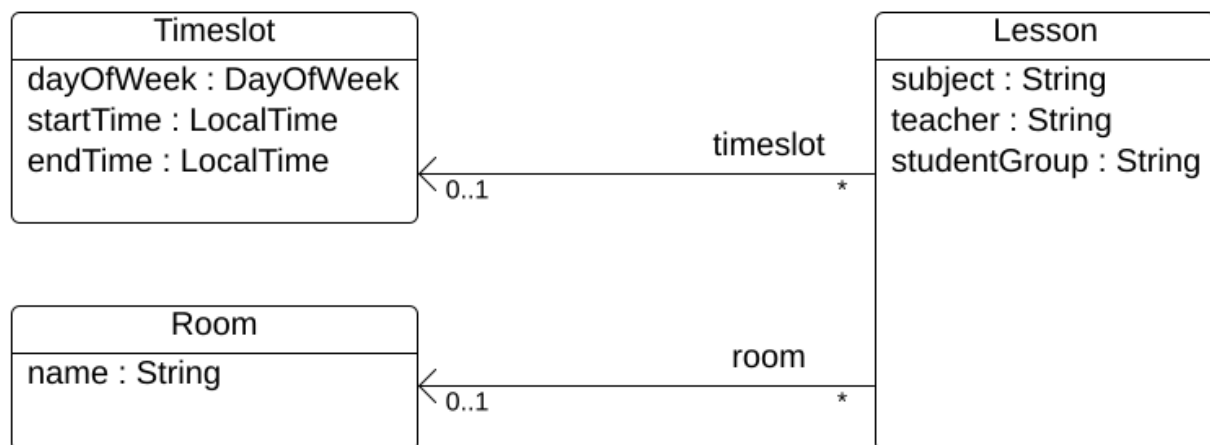
    <artifactId>optaplanner-quarkus</artifactId>
  </dependency>
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-quarkus-jackson</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-junit5</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

```

## 10.2. ドメインオブジェクトのモデル化

Red Hat ビルドの OptaPlanner の時間割プロジェクトの目標は、レッスンごとに時間枠と部屋に割り当てることです。これには、次の図に示すように、**Timeslot**、**Lesson**、および **Room** の3つのクラスを追加します。

### Time table class diagram



#### Timeslot

**Timeslot** クラスは、**Monday 10:30 - 11:30**、**Tuesday 13:30 - 14:30** など、授業の長さを表します。この例では、時間枠はすべて同じ長さ (期間) で、昼休みまたは他の休憩時間にはこのスロットはありません。

高校のスケジュールは毎週 (同じ内容が) 繰り返されるだけなので、時間枠には日付がありません。また、[継続的プランニング](#) は必要ありません。解決時に **Timeslot** インスタンスが変更しないため、**Timeslot** は **問題ファクト** と呼ばれます。このようなクラスには OptaPlanner 固有のアノテーションは必要ありません。

#### Room

**Room** クラスは、**Room A**、**Room B** など、授業の場所を表します。以下の例では、どの部屋も定員制限がなく、すべての授業に対応できます。

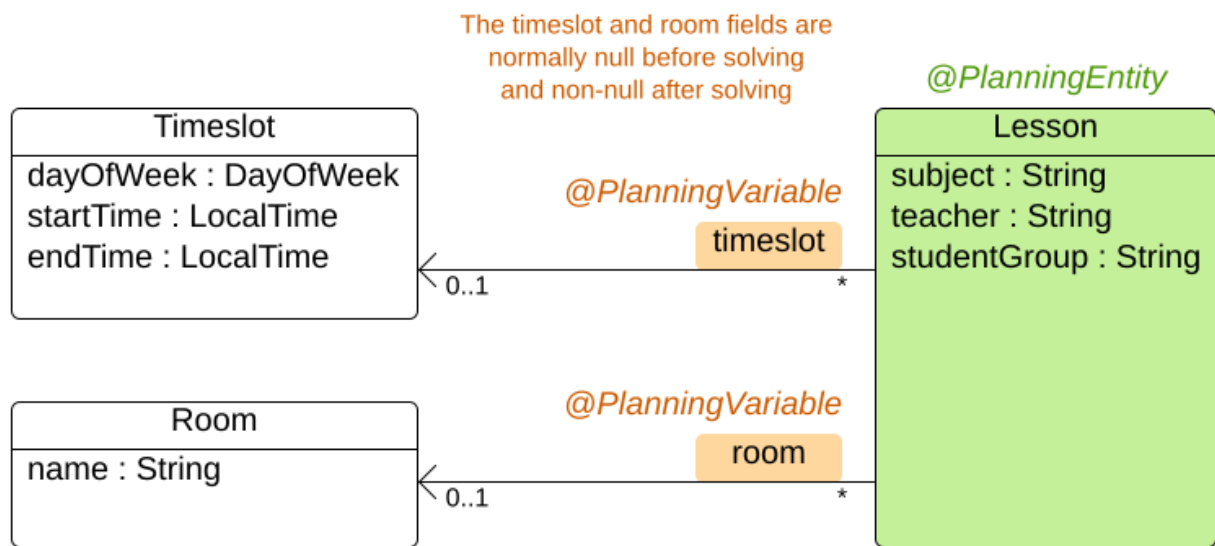
**Room** インスタンスは解決時に変化しないため、**Room** は **問題ファクト** でもあります。

## Lesson

授業中 (**Lesson** クラスで表現)、教師は複数の生徒に **Math by A.Turing for 9th grade**、**Chemistry by M.Curie for 10th grade** などの教科を指導します。ある教科について、毎週複数回、同じ教師が同じ生徒グループを指導する場合は、**Lesson** インスタンスが複数使用されますが、それらは **id** で識別可能です。たとえば、9 年生の場合は、1 週間に 6 回数学の授業があります。

解決中に、OptaPlanner は、**Lesson** クラスの **timeslot** フィールドと **room** フィールドを変更して、各授業を、時間枠 1 つ、部屋 1 つに割り当てます。OptaPlanner はこれらのフィールドを変更するため、**Lesson** は **プランニングエンティティ** となります。

## Time table class diagram



前図では、オレンジのフィールド以外のほぼすべてのフィールドに、入力データが含まれています。授業の **timeslot** フィールドと **room** フィールドは、入力データに割り当てられておらず (**null**)、出力データに割り当てられて (**null** ではない) います。Red Hat ビルドの OptaPlanner は、解決時にこれらのフィールドを変更します。このようなフィールドはプランニング変数と呼ばれます。このフィールドを OptaPlanner に認識させるには、**timeslot** フィールドと **room** のフィールドに **@PlanningVariable** アノテーションが必要です。このフィールドに含まれる **Lesson** クラスには、**@PlanningEntity** アノテーションが必要です。

## 手順

1. **src/main/java/com/example/domain/Timeslot.java** クラスを作成します。

```

package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;
  
```

```

private Timeslot() {
}

public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
    this.dayOfWeek = dayOfWeek;
    this.startTime = startTime;
    this.endTime = endTime;
}

@Override
public String toString() {
    return dayOfWeek + " " + startTime.toString();
}

// *****
// Getters and setters
// *****

public DayOfWeek getDayOfWeek() {
    return dayOfWeek;
}

public LocalTime getStartTime() {
    return startTime;
}

public LocalTime getEndTime() {
    return endTime;
}
}

```

後述しているように、**toString()** メソッドで出力を短くするため、OptaPlanner の **DEBUG** ログまたは **TRACE** ログの読み取りが簡単になっています。

2. **src/main/java/com/example/domain/Room.java** クラスを作成します。

```

package com.example.domain;

public class Room {

    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }
}

```

```

// *****
// Getters and setters
// *****

public String getName() {
    return name;
}

}

```

3. **src/main/java/com/example/domain/Lesson.java** クラスを作成します。

```

package com.example.domain;

import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // *****
    // Getters and setters
    // *****

    public Long getId() {
        return id;
    }

    public String getSubject() {

```

```

        return subject;
    }

    public String getTeacher() {
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
        return timeslot;
    }

    public void setTimeslot(Timeslot timeslot) {
        this.timeslot = timeslot;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }
}

```

**Lesson** クラスには **@PlanningEntity** アノテーションが含まれており、その中にプランニング変数が1つ以上含まれているため、OptaPlanner はこのクラスが解決時に変化することを認識します。

**timeslot** フィールドには **@PlanningVariable** アノテーションがあるため、OptaPlanner は、このフィールドの値が変化することを認識しています。このフィールドに割り当てることのできる **Timeslot** インスタンスを見つけ出すために、OptaPlanner は **valueRangeProviderRefs** プロパティを使用して値の範囲プロバイダーと連携し、**List<Timeslot>** を提供して選択できるようにします。値の範囲プロバイダーに関する詳細は、[「プランニングソリューションでのドメインオブジェクトの収集」](#) を参照してください。

**room** フィールドにも、同じ理由で **@PlanningVariable** アノテーションが含まれます。

### 10.3. 制約の定義およびスコアの計算

問題の解決時に **スコア** で導かれた解の質を表します。スコアが高いほど質が高くなります。Red Hat ビルドの OptaPlanner は、利用可能な時間内で見つかった解の中から最高スコアのものを探し出します。これが **最適** 解である可能性があります。

時間割の例のユースケースでは、ハードとソフト制約を使用しているため、**HardSoftScore** クラスでスコアを表します。

- ハード制約は、絶対に違反しないでください。たとえば、**部屋に同時に割り当てることができない授業は、最大1コマ**です。
- ソフト制約は、違反しないようにしてください。たとえば、**教師は、1つの部屋での授業を希望**します。

ハード制約は、他のハード制約と比べて、重み付けを行います。ソフト制約は、他のソフト制約と比べて、重み付けを行います。ハード制約は、それぞれの重みに関係なく、常にソフト制約よりも高くなります。

**EasyScoreCalculator** クラスを実装して、スコアを計算できます。

```
public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

    @Override
    public HardSoftScore calculateScore(TimeTable timeTable) {
        List<Lesson> lessonList = timeTable.getLessonList();
        int hardScore = 0;
        for (Lesson a : lessonList) {
            for (Lesson b : lessonList) {
                if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
                    && a.getId() < b.getId()) {
                    // A room can accommodate at most one lesson at the same time.
                    if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                        hardScore--;
                    }
                    // A teacher can teach at most one lesson at the same time.
                    if (a.getTeacher().equals(b.getTeacher())) {
                        hardScore--;
                    }
                    // A student can attend at most one lesson at the same time.
                    if (a.getStudentGroup().equals(b.getStudentGroup())) {
                        hardScore--;
                    }
                }
            }
        }
        int softScore = 0;
        // Soft constraints are only implemented in the "complete" implementation
        return HardSoftScore.of(hardScore, softScore);
    }
}
```

残念ながら、この解は漸増的ではないので、適切にスケーリングされません。授業が別の時間枠や教室に割り当てられるたびに、全授業が再評価され、新しいスコアが計算されます。

**src/main/java/com/example/solver/TimeTableConstraintProvider.java** クラスを作成して、漸増的スコア計算を実行すると、解がより優れたものになります。このクラスは、Java 8 Streams と SQL を基にした OptaPlanner の ConstraintStream API を使用します。**ConstraintProvider** は、**EasyScoreCalculator** と比べ、スケーリングの規模が遥かに大きくなっています ( $O(n^2)$  ではなく  $O(n)$ )。

## 手順

以下の **src/main/java/com/example/solver/TimeTableConstraintProvider.java** クラスを作成します。

```
package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
```

```

import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
            // Hard constraints
            roomConflict(constraintFactory),
            teacherConflict(constraintFactory),
            studentGroupConflict(constraintFactory),
            // Soft constraints are only implemented in the "complete" implementation
        };
    }

    private Constraint roomConflict(ConstraintFactory constraintFactory) {
        // A room can accommodate at most one lesson at the same time.

        // Select a lesson ...
        return constraintFactory.from(Lesson.class)
            // ... and pair it with another lesson ...
            .join(Lesson.class,
                // ... in the same timeslot ...
                Joiners.equal(Lesson::getTimeslot),
                // ... in the same room ...
                Joiners.equal(Lesson::getRoom),
                // ... and the pair is unique (different id, no reverse pairs)
                Joiners.lessThan(Lesson::getId))
            // then penalize each pair with a hard weight.
            .penalize("Room conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint teacherConflict(ConstraintFactory constraintFactory) {
        // A teacher can teach at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
            .join(Lesson.class,
                Joiners.equal(Lesson::getTimeslot),
                Joiners.equal(Lesson::getTeacher),
                Joiners.lessThan(Lesson::getId))
            .penalize("Teacher conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
        // A student can attend at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
            .join(Lesson.class,
                Joiners.equal(Lesson::getTimeslot),
                Joiners.equal(Lesson::getStudentGroup),
                Joiners.lessThan(Lesson::getId))
            .penalize("Student group conflict", HardSoftScore.ONE_HARD);
    }
}

```

## 10.4. プランニングソリューションでのドメインオブジェクトの収集

**TimeTable** インスタンスは、単一データセットの **Timeslot** インスタンス、**Room** インスタンス、および **Lesson** インスタンスをラップします。さらに、このインスタンスは、特定のプランニング変数の状態を持つ授業がすべて含まれているため、このインスタンスは **プランニングソリューション** となり、スコアが割り当てられます。

- 授業がまだ割り当てられていない場合は、スコアが **-4init/0hard/0soft** のソリューションなど、**初期化されていない** ソリューションとなります。
- ハード制約に違反する場合、スコアが **-2hard/-3soft** のソリューションなど、**実行不可** なソリューションとなります。
- 全ハード制約に準拠している場合は、スコアが **0hard/-7soft** など、**実行可能** なソリューションとなります。

**TimeTable** クラスには **@PlanningSolution** アノテーションが含まれているため、Red Hat ビルドの OptaPlanner はこのクラスに全入出力データが含まれていることを認識します。

具体的には、このクラスは問題の入力です。

- 全時間枠が含まれる **timeslotList** フィールド
  - これは、解決時に変更されないため、問題ファクトリリストです。
- 全部屋が含まれる **roomList** フィールド
  - これは、解決時に変更されないため、問題ファクトリリストです。
- 全授業が含まれる **lessonList** フィールド
  - これは、解決時に変更されるため、プランニングエンティティです。
  - 各 **Lesson**:
    - **timeslot** フィールドおよび **room** フィールドの値は通常、**null** で未割り当てです。これらの値は、プランニング変数です。
    - **subject**、**teacher**、**studentGroup** などの他のフィールドは入力されます。これらのフィールドは問題プロパティです。

ただし、このクラスはソリューションの出力でもあります。

- **Lesson** インスタンスごとの **lessonList** フィールドには、解決後は **null** ではない **timeslot** フィールドと **room** フィールドが含まれます。
- 出力ソリューションの品質を表す **score** フィールド (例: **0hard/-5soft**)

### 手順

**src/main/java/com/example/domain/TimeTable.java** クラスを作成します。

```
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
```



```

import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {

    @ValueRangeProvider(id = "timeslotRange")
    @ProblemFactCollectionProperty
    private List<Timeslot> timeslotList;

    @ValueRangeProvider(id = "roomRange")
    @ProblemFactCollectionProperty
    private List<Room> roomList;

    @PlanningEntityCollectionProperty
    private List<Lesson> lessonList;

    @PlanningScore
    private HardSoftScore score;

    private TimeTable() {
    }

    public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
        List<Lesson> lessonList) {
        this.timeslotList = timeslotList;
        this.roomList = roomList;
        this.lessonList = lessonList;
    }

    // *****
    // Getters and setters
    // *****

    public List<Timeslot> getTimeslotList() {
        return timeslotList;
    }

    public List<Room> getRoomList() {
        return roomList;
    }

    public List<Lesson> getLessonList() {
        return lessonList;
    }

    public HardSoftScore getScore() {
        return score;
    }

}

```

## 値の範囲のプロバイダー

**timeslotList** フィールドは、値の範囲プロバイダーです。これは **Timeslot** インスタンスを保持し、OptaPlanner がこのインスタンスを選択して、**Lesson** インスタンスの **timeslot** フィールドに割り当てることができます。**timeslotList** フィールドには **@ValueRangeProvider** アノテーションがあり、**id** を、**Lesson** の **@PlanningVariable** の **valueRangeProviderRefs** に一致させます。

同じロジックに従い、**roomList** フィールドにも **@ValueRangeProvider** アノテーションが含まれています。

### 問題ファクトとプランニングエンティティのプロパティ

さらに OptaPlanner は、変更可能な **Lesson** インスタンス、さらに **TimeTableConstraintProvider** によるスコア計算に使用する **Timeslot** インスタンスと **Room** インスタンスを取得する方法を把握しておく必要があります。

**timeslotList** フィールドと **roomList** フィールドには **@ProblemFactCollectionProperty** アノテーションが含まれているため、**TimeTableConstraintProvider** はこれらのインスタンスから選択できます。

**lessonList** には **@PlanningEntityCollectionProperty** アノテーションが含まれているため、OptaPlanner は解決時に変更でき、**TimeTableConstraintProvider** はこの中から選択できます。

## 10.5. SOLVER サービスの作成

REST スレッドで計画問題を解決すると、HTTP タイムアウトの問題が発生します。そのため、Quarkus スターターでは SolverManager を注入することで、個別のスレッドプールでソルバーを実行して複数のデータセットを並行して解決できます。

### 手順

**src/main/java/org/acme/optaplanner/rest/TimeTableResource.java** クラスを作成します。

```
package org.acme.optaplanner.rest;

import java.util.UUID;
import java.util.concurrent.ExecutionException;
import javax.inject.Inject;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

import org.acme.optaplanner.domain.TimeTable;
import org.optaplanner.core.api.solver.SolverJob;
import org.optaplanner.core.api.solver.SolverManager;

@Path("/timeTable")
public class TimeTableResource {

    @Inject
    SolverManager<TimeTable, UUID> solverManager;

    @POST
    @Path("/solve")
    public TimeTable solve(TimeTable problem) {
        UUID problemId = UUID.randomUUID();
        // Submit the problem to start solving
        SolverJob<TimeTable, UUID> solverJob = solverManager.solve(problemId, problem);
        TimeTable solution;
        try {
```

```

        // Wait until the solving ends
        solution = solverJob.getFinalBestSolution();
    } catch (InterruptedException | ExecutionException e) {
        throw new IllegalStateException("Solving failed.", e);
    }
    return solution;
}
}

```

この例では、初期実装はソルバーが完了するのを待つため、HTTP タイムアウトがまだ発生します。complete 実装を使用することで、より適切に HTTP タイムアウトを回避できます。

## 10.6. ソルバー終了時間の設定

プランニングアプリケーションに終了設定または終了イベントがない場合、理論的には永続的に実行されることになり、実際には HTTP タイムアウトエラーが発生します。これが発生しないようにするには、**optaplanner.solver.termination.spent-limit** パラメーターを使用して、アプリケーションが終了しからの時間を指定します。多くのアプリケーションでは、この時間を最低でも 5 分 (5m) に設定します。ただし、時間割の例では、解決時間を 5 分に制限すると、期間が十分に短いため HTTP タイムアウトを回避できます。

### 手順

以下の内容を含む **src/main/resources/application.properties** ファイルを作成します。

```
quarkus.optaplanner.solver.termination.spent-limit=5s
```

## 10.7. 時間割アプリケーションの実行

時間割プロジェクトを作成したら、開発モードで実行します。開発モードでは、アプリケーションの実行中にアプリケーションソースおよび設定を更新できます。変更が実行中のアプリケーションに反映されます。

### 前提条件

- 時間割プロジェクトを作成している。

### 手順

1. 開発モードでアプリケーションをコンパイルするには、プロジェクトディレクトリーから以下のコマンドを入力します。

```
./mvnw compile quarkus:dev
```

2. REST サービスをテストします。任意の REST クライアントを使用できます。この例では Linux **curl** コマンドを使用して POST 要求を送信します。

```
$ curl -i -X POST http://localhost:8080/timeTable/solve -H "Content-Type:application/json" -d
'{"timeslotList":[{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"}],"roomList":
[{"name":"Room A"}, {"name":"Room B"}], "lessonList":[{"id":1, "subject":"Math", "teacher":"A.
Turing", "studentGroup":"9th grade"}, {"id":2, "subject":"Chemistry", "teacher":"M.
```

```
Curie","studentGroup":"9th grade"},"id":3,"subject":"French","teacher":"M.
Curie","studentGroup":"10th grade"},"id":4,"subject":"History","teacher":"I.
Jones","studentGroup":"10th grade"]}]'
```

**application.properties** ファイルで定義した **終了時間** で指定した期間後に、サービスにより、以下の例のような出力が返されます。

```
HTTP/1.1 200
Content-Type: application/json
...

{"timeslotList":..., "roomList":..., "lessonList": [{"id":1, "subject":"Math", "teacher":"A.
Turing", "studentGroup":"9th grade", "timeslot":
{"dayOfWeek":"MONDAY", "startTime":"08:30:00", "endTime":"09:30:00"}, "room":
{"name":"Room A"}}, {"id":2, "subject":"Chemistry", "teacher":"M. Curie", "studentGroup":"9th
grade", "timeslot":
{"dayOfWeek":"MONDAY", "startTime":"09:30:00", "endTime":"10:30:00"}, "room":
{"name":"Room A"}}, {"id":3, "subject":"French", "teacher":"M. Curie", "studentGroup":"10th
grade", "timeslot":
{"dayOfWeek":"MONDAY", "startTime":"08:30:00", "endTime":"09:30:00"}, "room":
{"name":"Room B"}}, {"id":4, "subject":"History", "teacher":"I. Jones", "studentGroup":"10th
grade", "timeslot":
{"dayOfWeek":"MONDAY", "startTime":"09:30:00", "endTime":"10:30:00"}, "room":
{"name":"Room B"}}, {"score":"0hard/0soft"}
```

アプリケーションにより、4つの授業がすべて2つの時間枠、そして2つある部屋のうちの1つに割り当てられている点に注目してください。また、すべてのハード制約に準拠することに注意してください。たとえば、M. Curie の2つの授業は異なる時間スロットにあります。

3. 解決時間の OptaPlanner の実行内容を確認するには、サーバー側で情報ログを確認します。以下は、情報ログ出力の例です。

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score
calculation speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score
calculation speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed
(28524/sec), phase total (2), environment mode (REPRODUCIBLE).
```

## 10.8. アプリケーションのテスト

適切なアプリケーションにはテストが含まれます。timetable プロジェクトで制約とソルバーをテストします。

### 10.8.1. 学校の時間割の制約をテストする

timetable プロジェクトの各制約を個別にテストするには、単体テストで **ConstraintVerifier** を使用します。これにより、各制約のコーナーケースが他のテストから分離されてテストされるため、適切なテストカバレッジで新しい制約を追加する際のメンテナンスが軽減されます。

このテストは、制約 **TimeTableConstraintProvider::roomConflict** が、同じ部屋で3つのレッスンを与えられ、そのうちの2つのレッスンが同じタイムスロットを持つ場合、一致の重み1でペナルティを課すことを検証します。したがって、制約の重みが **10hard** の場合、スコアは **-10hard** 減少します。

## 手順

**src/test/java/org/acme/optaplanner/solver/TimeTableConstraintProviderTest.java** クラスを作成します。

```
package org.acme.optaplanner.solver;

import java.time.DayOfWeek;
import java.time.LocalTime;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.Room;
import org.acme.optaplanner.domain.TimeTable;
import org.acme.optaplanner.domain.Timeslot;
import org.junit.jupiter.api.Test;
import org.optaplanner.test.api.score.stream.ConstraintVerifier;

@QuarkusTest
class TimeTableConstraintProviderTest {

    private static final Room ROOM = new Room("Room1");
    private static final Timeslot TIMESLOT1 = new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9,0),
LocalTime.NOON);
    private static final Timeslot TIMESLOT2 = new Timeslot(DayOfWeek.TUESDAY,
LocalTime.of(9,0), LocalTime.NOON);

    @Inject
    ConstraintVerifier<TimeTableConstraintProvider, TimeTable> constraintVerifier;

    @Test
    void roomConflict() {
        Lesson firstLesson = new Lesson(1, "Subject1", "Teacher1", "Group1");
        Lesson conflictingLesson = new Lesson(2, "Subject2", "Teacher2", "Group2");
        Lesson nonConflictingLesson = new Lesson(3, "Subject3", "Teacher3", "Group3");

        firstLesson.setRoom(ROOM);
        firstLesson.setTimeslot(TIMESLOT1);

        conflictingLesson.setRoom(ROOM);
        conflictingLesson.setTimeslot(TIMESLOT1);

        nonConflictingLesson.setRoom(ROOM);
        nonConflictingLesson.setTimeslot(TIMESLOT2);

        constraintVerifier.verifyThat(TimeTableConstraintProvider::roomConflict)
            .given(firstLesson, conflictingLesson, nonConflictingLesson)
            .penalizesBy(1);
    }
}
```

}

}

制約の重みが **ConstraintProvider** でハードコーディングされている場合でも、**ConstraintVerifier** がテスト中に制約の重みを無視することに注意してください。これは、実稼動に入る前に制約の重みが定期的に変更されるためです。このように、制約の重みの微調整によって単体テストが中断されることはありません。

### 10.8.2. 学校の時間割ソルバーをテストする

以下の例では、Red Hat ビルドの Quarkus で Red Hat ビルドの OptaPlanner の時間割プロジェクトをテストします。このアプリケーションは、JUnit テストを使用してテストのデータセットを生成し、**TimeTableController** に送信して解決します。

#### 手順

1. 以下の内容を含む **src/test/java/com/example/rest/TimeTableResourceTest.java** クラスを作成します。

```
package com.exmaple.optaplanner.rest;

import java.time.DayOfWeek;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import com.exmaple.optaplanner.domain.Room;
import com.exmaple.optaplanner.domain.Timeslot;
import com.exmaple.optaplanner.domain.Lesson;
import com.exmaple.optaplanner.domain.TimeTable;
import com.exmaple.optaplanner.rest.TimeTableResource;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableResource.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
```

```

        assertNotNull(lesson.getTimeslot());
        assertNotNull(lesson.getRoom());
    }
    assertTrue(solution.getScore().isFeasible());
}

private TimeTable generateProblem() {
    List<Timeslot> timeslotList = new ArrayList<>();
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
    LocalTime.of(9, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
    LocalTime.of(10, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
    LocalTime.of(11, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
    LocalTime.of(14, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
    LocalTime.of(15, 30)));

    List<Room> roomList = new ArrayList<>();
    roomList.add(new Room("Room A"));
    roomList.add(new Room("Room B"));
    roomList.add(new Room("Room C"));

    List<Lesson> lessonList = new ArrayList<>();
    lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
    lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
    lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
    lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
    lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

    lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
    lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
    lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
    lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
    lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
    return new TimeTable(timeslotList, roomList, lessonList);
}
}

```

このテストは、解決後にすべての授業がタイムスロットと部屋に割り当てられていることを確認します。また、実行可能解（ハード制約の違反なし）も確認します。

2. テストプロパティを **src / main / resources / application.properties** ファイルに追加します。

```

# The solver runs only for 5 seconds to avoid a HTTP timeout in this simple implementation.
# It's recommended to run for at least 5 minutes ("5m") otherwise.
quarkus.optaplanner.solver.termination.spent-limit=5s

# Effectively disable this termination in favor of the best-score-limit
%test.quarkus.optaplanner.solver.termination.spent-limit=1h
%test.quarkus.optaplanner.solver.termination.best-score-limit=0hard/*soft

```

通常、ソルバーは 200 ミリ秒未満で実行可能解を検索します。**application.properties** が、実行可能な

ソリューション (**0hard/\*soft**) が見つかりと同時に終了するように、テスト中のソルバーの終了を上書きします。こうすることで、ユニットテストが任意のハードウェアで実行される可能性があるため、ソルバーの時間をハードコード化するのを回避します。このアプローチを使用することで、動きが遅いシステムであっても、実行可能なソリューションを検索するのに十分な時間だけテストが実行されます。ただし、高速システムでも、厳密に必要とされる時間よりもミリ秒単位で長く実行されることはありません。

## 10.9. ロギング

Red Hat ビルドの OptaPlanner の時間割プロジェクトを完了後にロギング情報を使用すると、**ConstraintProvider** で制約が微調整しやすくなります。**info** ログファイルでスコア計算の速度を確認して、制約に加えた変更の影響を評価します。デバッグモードでアプリケーションを実行して、アプリケーションが行う手順をすべて表示するか、追跡ログを使用して全手順および動きをロギングします。

### 手順

1. 時間割アプリケーションを一定の時間 (例: 5 分) 実行します。
2. 以下の例のように、**log** ファイルのスコア計算の速度を確認します。

```
... Solving ended: ..., score calculation speed (29455/sec), ...
```

3. 制約を変更して、同じ時間、プランニングアプリケーションを実行し、**log** ファイルに記録されているスコア計算速度を確認します。
4. アプリケーションをデバッグモードで実行して、アプリケーションの全実行ステップをログに記録します。
  - コマンドラインからデバッグモードを実行するには、**-D** システムプロパティを使用します。
  - デバッグモードを永続的に有効にするには、以下の行を **application.properties** ファイルに追加します。

```
quarkus.log.category."org.optaplanner".level=debug
```

以下の例では、デバッグモードでの **log** ファイルの出力を表示します。

```
... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode (REPRODUCIBLE), random (JDK with seed 0).
... CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15),
picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
... CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15),
picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
...
```

5. **trace** ロギングを使用して、全手順、および手順ごとの全動きを表示します。

## 10.10. データベースを QUARKUS OPTAPLANNER 学校の時間割アプリケーションと統合する

Quarkus OptaPlanner 学校の時間割アプリケーションを作成したら、それをデータベースと統合し、ウェブベースのユーザーインターフェイスを作成して時間割を表示できます。



## 前提条件

- Quarkus OptaPlanner 学校の時間割アプリケーションがあります。

## 手順

1. Hibernate と Panache を使用して、**Timeslot**、**Room**、および **Lesson** インスタンスをデータベースに格納します。詳細については、[Panache を使用した単純化された Hibernate ORM](#) を参照してください。
2. REST を介してインスタンスを公開します。詳細については、[JSON REST サービスの記述](#) を参照してください。
3. **TimeTableResource** クラスを更新して、単一のトランザクションで **TimeTable** インスタンスの読み取りと書き込みを行います。

```
package org.acme.optaplanner.rest;

import javax.inject.Inject;
import javax.transaction.Transactional;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

import io.quarkus.panache.common.Sort;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.Room;
import org.acme.optaplanner.domain.TimeTable;
import org.acme.optaplanner.domain.Timeslot;
import org.optaplanner.core.api.score.ScoreManager;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.solver.SolverManager;
import org.optaplanner.core.api.solver.SolverStatus;

@Path("/timeTable")
public class TimeTableResource {

    public static final Long SINGLETON_TIME_TABLE_ID = 1L;

    @Inject
    SolverManager<TimeTable, Long> solverManager;
    @Inject
    ScoreManager<TimeTable, HardSoftScore> scoreManager;

    // To try, open http://localhost:8080/timeTable
    @GET
    public TimeTable getTimeTable() {
        // Get the solver status before loading the solution
        // to avoid the race condition that the solver terminates between them
        SolverStatus solverStatus = getSolverStatus();
        TimeTable solution = findById(SINGLETON_TIME_TABLE_ID);
        scoreManager.updateScore(solution); // Sets the score
        solution.setSolverStatus(solverStatus);
        return solution;
    }
}
```

```

@POST
@Path("/solve")
public void solve() {
    solverManager.solveAndListen(SINGLETON_TIME_TABLE_ID,
        this::findById,
        this::save);
}

public SolverStatus getSolverStatus() {
    return solverManager.getSolverStatus(SINGLETON_TIME_TABLE_ID);
}

@POST
@Path("/stopSolving")
public void stopSolving() {
    solverManager.terminateEarly(SINGLETON_TIME_TABLE_ID);
}

@Transactional
protected TimeTable findById(Long id) {
    if (!SINGLETON_TIME_TABLE_ID.equals(id)) {
        throw new IllegalStateException("There is no timeTable with id (" + id + ").");
    }
    // Occurs in a single transaction, so each initialized lesson references the same
    timeslot/room instance
    // that is contained by the timeTable's timeslotList/roomList.
    return new TimeTable(
        Timeslot.listAll(Sort.by("dayOfWeek").and("startTime").and("endTime").and("id")),
        Room.listAll(Sort.by("name").and("id")),
        Lesson.listAll(Sort.by("subject").and("teacher").and("studentGroup").and("id")));
}

@Transactional
protected void save(TimeTable timeTable) {
    for (Lesson lesson : timeTable.getLessonList()) {
        // TODO this is awfully naive: optimistic locking causes issues if called by the
        SolverManager
        Lesson attachedLesson = Lesson.findById(lesson.getId());
        attachedLesson.setTimeslot(lesson.getTimeslot());
        attachedLesson.setRoom(lesson.getRoom());
    }
}
}

```

この例には **TimeTable** インスタンスが含まれています。ただし、マルチテナンシーを有効にして、複数の学校の **TimeTable** インスタンスを並行して処理できます。

**getTimeTable()** メソッドは、データベースから最新の時間割を返します。自動的に挿入される **ScoreManager** メソッドを使用して、そのタイムテーブルのスコアを計算し、UI で使用できるようにします。

**solve()** メソッドは、ジョブを開始して、現在の時間割を解決し、時間枠と部屋の割り当てをデータベースに保存します。このメソッドは、**SolverManager.solveAndListen()** メソッドを使用して、中間の最適解をリッスンし、それに合わせてデータベースを更新します。バックエンドがまだ解決している間、UI はこれを使用して進行状況を表示します。

4. **TimeTableResourceTest** クラスを更新して、**solve()** メソッドがすぐに戻ることを反映し、ソルバーが解決を完了するまで最新の解をポーリングするようにします。

```
package org.acme.optaplanner.rest;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.TimeTable;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.optaplanner.core.api.solver.SolverStatus;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

    @Test
    @Timeout(600_000)
    public void solveDemoDataUntilFeasible() throws InterruptedException {
        timeTableResource.solve();
        TimeTable timeTable = timeTableResource.getTimeTable();
        while (timeTable.getSolverStatus() != SolverStatus.NOT_SOLVING) {
            // Quick polling (not a Test Thread Sleep anti-pattern)
            // Test is still fast on fast machines and doesn't randomly fail on slow machines.
            Thread.sleep(20L);
            timeTable = timeTableResource.getTimeTable();
        }
        assertFalse(timeTable.getLessonList().isEmpty());
        for (Lesson lesson : timeTable.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(timeTable.getScore().isFeasible());
    }
}
```

5. これらの REST メソッドの上に Web UI を構築して、タイムテーブルを視覚的に表現します。
6. [クイックスタートソースコード](#)を確認します。

## 10.11. MICROMETER と PROMETHEUS を使用して学校の時間割を監視する OPTAPLANNER QUARKUS アプリケーション

OptaPlanner は、Java アプリケーション用のメトリック計測ライブラリーである [Micrometer](#) を介してメトリックを公開します。Prometheus で Micrometer を使用して、学校の時間割アプリケーションで OptaPlanner ソルバーを監視できます。

## 前提条件

- Quarkus OptaPlanner 学校の時間割アプリケーションを作成しました。
- Prometheus がインストールされている。Prometheus のインストールについては、[Prometheus](#) の Web サイトを参照してください。

## 手順

1. 学校の時間割 **pom.xml** ファイルに Micrometer Prometheus 依存関係を追加します。

```
<dependency>  
  <groupId>io.quarkus</groupId>  
  <artifactId>quarkus-micrometer-registry-prometheus</artifactId>  
</dependency>
```

2. 学校の時間割アプリケーションを開始します。

```
mvn compile quarkus:dev
```

3. Web ブラウザーで <http://localhost:8080/q/metric> を開きます。

## 第11章 RED HAT ビルドの QUARKUS での RED HAT ビルドの OPTAPLANNER: ワクチン接種予約スケジューラーのクイックスタートガイド

OptaPlanner ワクチン接種予約スケジューラーのクイックスタートを使用して、効率性および公平性の高いワクチン接種スケジュールを開発できます。ワクチン接種予約スケジューラーは、人工知能 (AI) を使用して摂取者の優先順位を決定し、複数の制約や優先順位に基づいて時間枠を割り当てます。

### 前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスマーポータル (ログインが必要) の [ソフトウェアダウンロード](#) ページから入手できます。
- Apache Maven 3.6 以降がインストールされている。Maven は [Apache Maven Project](#) の Web サイトから入手できます。
- IntelliJ IDEA、VSCode、Eclipse、NetBeans などの IDE が利用できる。
- [6章 OptaPlanner および Quarkus の使用ガイド](#) の説明に従って、Quarkus OptaPlanner プロジェクトを作成している。

### 11.1. OPTAPLANNER ワクチン接種予約のスケジューラーの仕組み

スケジュール予約には、主に 2 つの方法があります。システムは、ユーザーが予約枠 (ユーザーの選択) を選択できるようにするか、システムが予約枠を割り当ててそのユーザーにワクチン予約の日付と場所 (システムの自動割り当て) を通知することができます。OptaPlanner のワクチン接種予約スケジューラーは、システム自動割り当てのアプローチを使用します。OptaPlanner ワクチン接種予約スケジューラーでは、システムに情報を提供するアプリケーションを作成して、システムが予約を割り当てます。

このアプローチの特徴は次のとおりです。

- 予約枠は優先順位に基づいて割り当てられます。
- システムは、事前設定したプランニング制約に基づいて、最適な時間と場所を割り当てます。
- システムは、多数のユーザーが少ない予約数に殺到しても、圧迫されることはありません。

このアプローチにより、プランニング制約を使用して各ユーザーのスコアを作成し、できるだけ多くの人がワクチン接種を受けられるようにする問題を解決します。スコアで、予約を取得するタイミングが決まります。スコアが高いほど、早い段階で受信できる可能性が高くなります。

#### 11.1.1. OptaPlanner ワクチン接種予約のスケジューラーの制約

OptaPlanner ワクチン接種予約のスケジューラー制約は、ハード、中程度、またはソフトのいずれかです。

- ハード制約は、絶対に違反できません。ハード制約に違反している場合には、プランは実行不可なので実行できません。
  - 容量: 場所、時間に関わらず、ワクチン量よりも多く予約を取らない。
  - ワクチン接種年齢上限: ワクチンに年齢の上限がある場合には、1 回目のワクチン接種時にワクチンの年齢の上限以上の方には投与しないようにする。年齢にあったワクチンタイプを投与するようにします。たとえば、75 歳の方には、年齢の上限が 65 歳のワクチンの予

約を割り当てないようにします。

- 必要なワクチンタイプ: 必要なワクチンタイプを使用する。たとえば、2 回目のワクチンは、1 回目と同じタイプを使用する必要があります。
- 準備が整う日: 指定の日付以降にワクチンを投与する。たとえば、2 回目の投与を受ける場合に、特定のワクチンタイプを最短で接種が可能な推奨日時よりも前に投与しないでください (例: 1 回目の接種から 26 日後など)。
- 期日: 指定された日付以前にワクチンを接種する。たとえば、2 回目の投与を受ける場合に、特定のワクチンタイプを接種可能な推奨最終日よりも前に投与してください (例: 1 回目の接種から 3 ヶ月後など)。
- 移動の最大距離の制限: 最寄りのワクチンセンターグループに、各接種者を割り当てる。通常、これは 3 つのセンターのいずれかになります。この制限は、距離ではなく移動時間によって計算されるので、郊外に居住の方に比べ、都市部に居住する方は最大距離が短くなります。
- 中程度の制約は、全員に予約を割り当てる空きがない場合に、どの接種希望者が予約できないかを決定します。これは、過剰制約プランニングと呼ばれます。
  - 2 回目のワクチン接種予約: 理想の日付が計画枠外になってしまう場合を除き、2 回目のワクチン接種予約を割り当てずに放置しないようにする。
  - 優先度評価に基づくスケジュール設定: 各接種者に優先度評価がある。これは、通常年齢ですが、医療関係者などの場合には優先度が高くなります。優先度が最も低い場合にのみ、予約を割り当てずに放置できます。次のワクチン接種のタイミングで考慮されます。2 回目の投与は優先度評価よりも優先されるので、この制約は以前の制約よりもソフトです。
- ソフト制約は、違反しないようにしてください。
  - 希望のワクチンセンター: 接種者が希望するワクチンセンターがある場合は、そのセンターで予約を入れる。
  - 距離: 接種者が割り当てられたワクチンセンターまで移動する距離を最小限に抑える。
  - 理想の日付: できるだけ指定の日付に近い日にワクチンを投与する。たとえば、2 回目の投与を受ける場合に、特定のワクチンで理想の日付に投与します (例: 1 回目の接種から 28 日後など)。この制約は、距離制約よりもソフトで、理想の日付に近づけるために、遠方まで出向させる必要がないようにします。
  - 優先度評価: 優先度評価の高い方から、予約枠で日付に近い順に割り当てる。この制約は、距離制約よりもソフトで、遠方まで出向させる必要がないようにします。2 回目の投与は優先度評価よりも優先されるので、この制約も以前の制約よりもソフトです。

ハード制約は、他のハード制約と比べて、重み付けを行います。ソフト制約は、他のソフト制約と比べて、重み付けを行います。ただし、ハード制約は、常に中程度およびソフト制約よりも優先されます。ハード制約に違反している場合には、プランは実行できません。ただし、ハード制約に違反していない場合には、優先度を判定するためにソフト制約および中程度の制約が考慮されます。空きがある予約枠よりも希望者が多いので、優先付けが必要です。2 回目の投与予約は必ず先に割り当て、後ほどシステムに負荷がかからないようにバックログの作成は回避します。その後、優先度評価をもとに割り当てていきます。優先度評価年齢から評価を開始します。この評価では、若い人よりお年寄りが優先されます。その後、特定の優先度のグループには、200-300 ポイントなど、余分にポイントが追加されます。これはグループの優先順位によって異なります。たとえば、看護師は追加で 1000 ポイントを受け取る可能性があります。こうすることで、年齢の高い看護師は、若い看護師よりも優先され、若い看護師は看護師ではない人よりも優先されます。以下の表は、この概念を示しています。

表11.1 優先順位評価の表

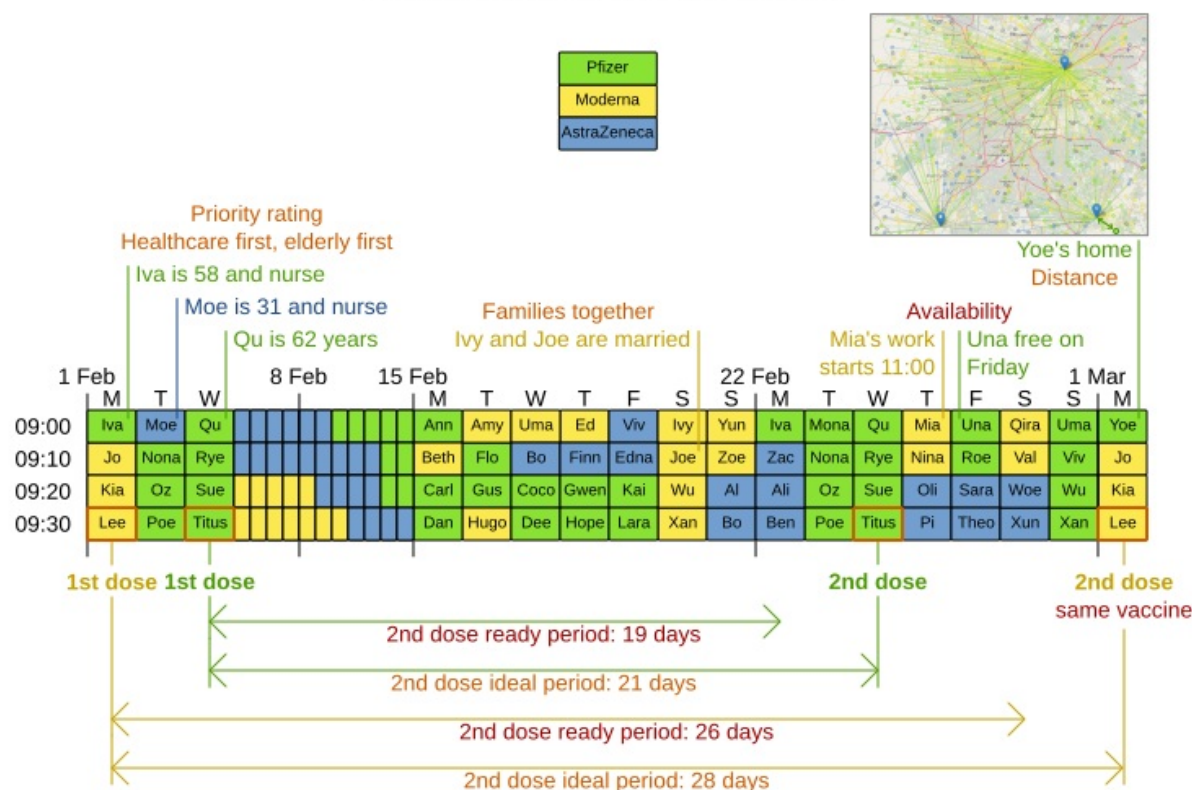
年齢	職業	優先順位の評価
60	看護師	1060
33	看護師	1033
71	定年退職	71
52	オフィスワーカー	52

### 11.1.2. OptaPlanner のソルバー

OptaPlanner のコアには Solver があり、エンジンは、問題データセットを取り、プランニングの制約および設定をオーバーレイします。問題データセットには、人、ワクチン、ワクチンセンターなどの情報すべてが含まれます。ソルバーは、さまざまなデータを組み合わせて機能し、最終的に特定のセンターでワクチン接種予約を割り当てる時に、最適な予約スケジュールを決定します。以下は、ソルバーで作成されたスケジュールを示しています。

## Vaccination scheduling

Assign people to vaccination appointments.



### 11.1.3. 継続プランニング

継続プランニングは、1つ以上の今後のプランニング期間をまとめて管理して、そのプロセスを月単位、週単位、日単位、1時間単位、またはそれよりも短い単位で繰り返す手法です。プランニング枠は、指定した間隔で増分して進められます。以下は、毎日更新される2週間のプランニング枠を示して



います。

## Vaccination scheduling: continuous planning



2週間計画枠を半分に分割します。1週間目は公開状態で、2週間目がドラフト状態にあります。プランニング枠の公開部分とドラフト部分の両方で、予約を割り当てます。ただし、プランニング枠の公開部分の方だけに、予約の通知が行きます。他の予約は、次の実行でまだ簡単に変更できます。こうすることで、ソルバーを次回実行すると、必要に応じて OptaPlanner が柔軟にドラフトの部分の予約を変更できます。たとえば、2回目の投与の準備が月曜にできており、理想の日付が水曜の場合には、同じ週のドラフト予約を指定できることが証明できるのであれば、OptaPlanner は月曜に予約を割り当てる必要はありません。

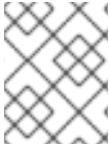
プランニング枠のサイズは指定できますが、問題領域のサイズを認識しておいてください。問題領域は、スケジュールの作成に使用されるさまざまな要素すべてです。計画期間が長いと、問題領域が大きくなります。

### 11.1.4. 固定されたプランニングエンティティ

1日単位で継続的にプランニングしている場合には、2週間の中ですでに割り当てられている予約枠があります。ダブルブッキングされないように、OptaPlanner は予約枠が割り当て済みと固定することができます。1つ以上の特有の割当のアンカリングや、OptaPlanner が強制的に固定の割当を除外してスケジュールするのに、固定機能を使用します。予約など、固定されたプランニングエンティティは、解決時には変更されません。

エンティティが固定されているかどうかは、予約の状態により分かります。予約の状態には **Open**、**Invited**、**Accepted**、**Rejected** または **Rescheduled** があります。





## 注記

OptaPlanner エンジンでは、予約が固定されているかどうかのみに着目するので、クイックスタートのデモコードに予約の状態は直接表示されません。

スケジュール済みの予約に近い日付で、プランニングできるようにする必要があります。状態が **Invited** または **Accepted** の予約が固定されます。状態が **Open**、**Reschedule** および **Rejected** の予約は固定せず、スケジュールリングに利用できます。

この例では、ソルバーが実行されると、公開範囲とドラフト範囲の両方を含めた 2 週間の計画枠全体を検索します。ソルバーは、スケジュールリング前の入力データに加え、固定されておらず、状態が **Open**、**Reschedule** または **Rejected** の予約、エンティティを検討して、最適解を見つけ出します。ソルバーを日次で実行する場合には、ソルバーを実行する前に新しい日付が追加されることが確認できます。

新しい日付に予約が割り当てられ、固定ウィンドウのドラフト部分で、これまでにスケジュールされていた Amy と Edna が公開部分の枠で予約が取れていることが分かります。これは、Gus および Hugo が再スケジュールを依頼したので可能となりました。Amy と Edna にはドラフトの日付が通知されていないので、混乱を招くことはありません。これで、計画枠の公開部分で予約が取れたので、Amy と Edna には通知が送信され、その予約を承諾するか拒否するかが確認され、この 2 人の予約が固定されます。

## 11.2. OPTAPLANNER ワクチン接種予約スケジューラーのダウンロードおよび実行

OptaPlanner ワクチン接種予約スケジューラークイックスタートをダウンロードし、Quarkus の開発モードで起動して、ブラウザでアプリケーションを表示します。Quarkus 開発モードを使用すると、実行中にアプリケーションを変更し、更新できます。

### 手順

1. Red Hat カスタマーポータル [Software Downloads](#) ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
  - Product: Decision Manager
  - Version: 7.12
2. Red Hat Decision Manager 7.12.0 Kogito および OptaPlanner 8 Decision Services クイックスタート (`rhdm-7.12.0-kogito-and-optaplanner-quickstarts.zip`) をダウンロードします。
3. `rhdm-7.12.0-kogito-and-optaplanner-quickstarts.zip` ファイルを展開します。
4. `optaplanner-quickstarts-8.11.1.Final-redhat-00006` ディレクトリーに移動します。
5. `optaplanner-quickstarts-8.11.1.Final-redhat-00006/use-cases/vaccination-scheduling` ディレクトリーに移動します。
6. 以下のコマンドを入力して、開発モードで OptaPlanner 解析スケジューラーを起動します。

```
$ mvn quarkus:dev
```

7. OptaPlanner ワクチン接種予約スケジューラーを表示するには、Web ブラウザーに以下の URL を入力します。

```
http://localhost:8080/
```

8. OptaPlanner ワクチン接種予約スケジューラーを実行するには、**Solve** をクリックします。
9. ソースコードに変更を加えてから、F5 キーを押してブラウザを更新します。加えた変更が有効になったことを確認してください。

### 11.3. OPTAPLANNER ワクチン接種予約スケジューラーのパッケージ化および実行

**quarkus:dev** モードの OptaPlanner ワクチン接種予約スケジューラーで開発作業が完了したら、アプリケーションを従来の jar ファイルとして実行します。

#### 前提条件

- OptaPlanner ワクチン接種予約スケジューラーのクイックスタートをダウンロードしている。詳細は、「[OptaPlanner ワクチン接種予約スケジューラーのダウンロードおよび実行](#)」を参照してください。

#### 手順

1. **/use-cases/vaccination-scheduling** ディレクトリーに移動します。
2. OptaPlanner 解析スケジューラーをコンパイルするには、以下のコマンドを入力します。

```
$ mvn package
```

3. コンパイルした OptaPlanner ワクチン接種予約スケジューラーを実行するには、以下のコマンドを入力します。

```
$ java -jar ./target/*-runner.jar
```



#### 注記

ポート 8081 でアプリケーションを実行するには、**-Dquarkus.http.port=8081** を前述のコマンドに追加します。

4. OptaPlanner ワクチン接種予約スケジューラーを起動するには、Web ブラウザーに以下の URL を入力します。

```
http://localhost:8080/
```

### 11.4. OPTAPLANNER ワクチン接種予約スケジューラーのネイティブ実行可能ファイルとしての実行

Quarkus が提供するサイズの小さいメモリーフットプリントや、アクセス速度を活用し、OptaPlanner ワクチン接種予約スケジューラーを Quarkus ネイティブモードでコンパイルします。

#### 手順

1. GraalVM と **native-image** ツールをインストールします。詳細は、Quarkus Web サイトの [Configuring GraalVM](#) を参照してください。
2. `/use-cases/vaccination-scheduling` ディレクトリーに移動します。
3. OptaPlanner 解析スケジューラーをネイティブにコンパイルするには、以下のコマンドを入力します。

```
$ mvn package -Dnative -DskipTests
```

4. ネイティブ実行可能ファイルを実行するには、以下のコマンドを入力します。

```
$ ./target/*-runner
```

5. OptaPlanner ワクチン接種予約スケジューラーを起動するには、Web ブラウザーに以下の URL を入力します。

```
http://localhost:8080/
```

## 11.5. 関連情報

- [スケジュールスケジュール動画 \(英語版\)](#)

## 第12章 SPRING BOOT 上の RED HAT ビルドの OPTAPLANNER: 時間割のクイックスタートガイド

本書では、OptaPlanner の制約解決人工知能 (AI) を使用して Spring Boot アプリケーションを作成するプロセスを説明します。生徒と教師の時間割を最適化する REST アプリケーションを構築します。

<a href="#">Refresh</a>	<a href="#">▶ Solve</a>	Score: 0hard/18soft	<a href="#">By room</a>	<a href="#">By teacher</a>	<a href="#">By student group</a>
Timeslot	Room A	Room B	Room C		
Monday 08:30 - 09:30		Physics by M. Curie 10th grade 27	Spanish by P. Cruz 9th grade 22		
Monday 09:30 - 10:30		Physics by M. Curie 9th grade 16	Spanish by P. Cruz 10th grade 33		
Monday 10:30 - 11:30	Geography by C. Darwin 10th grade 30	Chemistry by M. Curie 9th grade 17			
Monday 13:30 - 14:30		Math by A. Turing 10th grade 26	English by I. Jones 9th grade 20		
Monday 14:30 - 15:30		Math by A. Turing 10th grade 25	English by I. Jones 9th grade 21		

サービスは、AI を使用して、以下のハードおよびソフトの **スケジュール制約** に準拠し、**Lesson** インスタンスを **Timeslot** インスタンスと **Room** インスタンスに自動的に割り当てます。

- 1部屋に同時に割り当てることができる授業は、最大1コマです。
- 教師が同時に一度に行うことができる授業は最大1回です。
- 生徒は同時に出席できる授業は最大1コマです。
- 教師は、1つの部屋での授業を希望します。
- 教師は、連続した授業を好み、授業間に時間が空くの嫌います。

数学的に考えると、学校の時間割は **NP 困難** の問題であります。つまり、スケーリングが困難です。総当たり攻撃で考えられる組み合わせを単純にすべて反復すると、スーパーコンピュータを使用したとしても、非自明的なデータセットを取得するのに数百年かかります。幸い、OptaPlanner などの AI 制約ソルバーには、妥当な時間内にほぼ最適なソリューションを提供する高度なアルゴリズムがあります。妥当な期間として考慮される内容は、問題の目的によって異なります。

### 前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスタマーポータル (ログインが必要) の [ソフトウェアダウンロード](#) ページから入手できます。

- Apache Maven 3.6 以降がインストールされている。Maven は [Apache Maven Project](#) の Web サイトから入手できます。
- IntelliJ IDEA、VSCode、Eclipse、NetBeans などの IDE が利用できる。

## 12.1. SPRING BOOT 時間割のクイックスタートのダウンロードおよびビルド

Spring Boot 製品を備えた Red Hat ビルドの OptaPlanner 向けの授業の時間割プロジェクトの完全な例を表示するには、Red Hat カスタマーポータルからスターターアプリケーションをダウンロードします。

### 手順

1. Red Hat カスタマーポータルの [Software Downloads](#) ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
  - Product: Decision Manager
  - Version: 7.12
2. Red Hat Decision Manager 7.12.0 Kogito および OptaPlanner 8 Decision Services クイックスタート (**rhdm-7.12.0-kogito-and-optaplanner-quickstarts.zip**) をダウンロードします。
3. **rhdm-7.12.0-kogito-and-optaplanner-quickstarts.zip** ファイルを展開します。
4. Red Hat Decision Manager 7.12.0 Kogito および OptaPlanner 8 Decision Services Maven Repository (**rhdm-7.12.0-kogito-maven-repository.zip**) を **ダウンロード** します。
5. **rhdm-7.12.0-kogito-maven-repository.zip** ファイルを抽出します。
6. **rhdm-7.12.0-kogito-maven-repository / maven-repository** サブディレクトリーの内容を **~/ .m2 / repository** ディレクトリーにコピーします。
7. **optaplanner-quickstarts-8.11.1.Final-redhat-00006/technology/java-spring-boot** ディレクトリーに移動します。
8. 以下のコマンドを入力して、Spring Boot の授業の時間割プロジェクトをビルドします。

```
mvn clean install -DskipTests
```

9. Spring Boot の授業の時間割プロジェクトをビルドするには、以下のコマンドを入力します。

```
mvn spring-boot:run -DskipTests
```

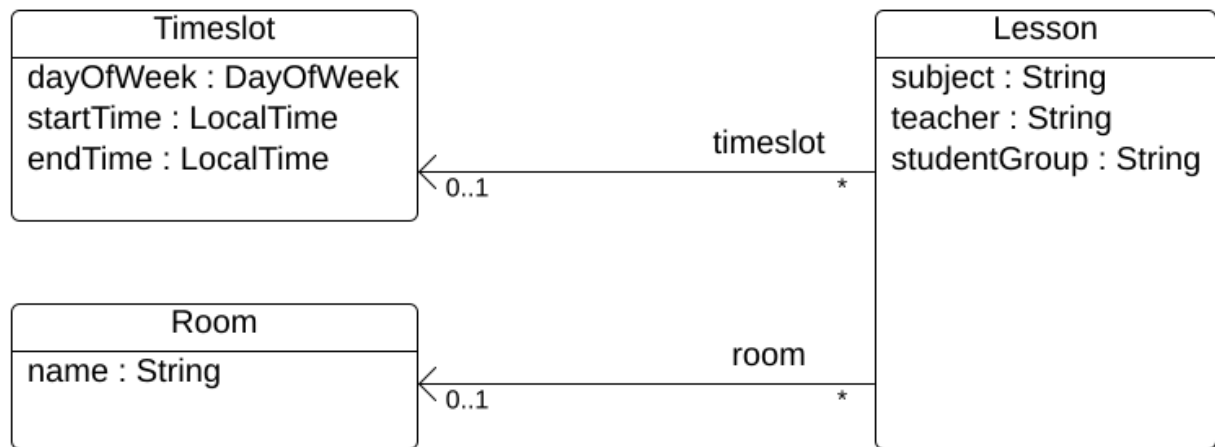
10. プロジェクトを表示するには、Web ブラウザーで以下の URL を入力します。

```
http://localhost:8080/
```

## 12.2. ドメインオブジェクトのモデル化

Red Hat ビルドの OptaPlanner の時間割プロジェクトの目標は、レッスンごとに時間枠と部屋に割り当てることです。これには、次の図に示すように、**Timeslot**、**Lesson**、および **Room** の 3 つのクラスを追加します。

## Time table class diagram



### Timeslot

**Timeslot** クラスは、**Monday 10:30 - 11:30**、**Tuesday 13:30 - 14:30** など、授業の長さを表します。この例では、時間枠はすべて同じ長さ (期間) で、昼休みまたは他の休憩時間にはこのスロットはありません。

高校のスケジュールは毎週 (同じ内容が) 繰り返されるだけなので、時間枠には日付がありません。また、[継続的プランニング](#) は必要ありません。解決時に **Timeslot** インスタンスが変更しないため、Timeslot は **問題ファクト** と呼ばれます。このようなクラスには OptaPlanner 固有のアノテーションは必要ありません。

### Room

**Room** クラスは、**Room A**、**Room B** など、授業の場所を表します。以下の例では、どの部屋も定員制限がなく、すべての授業に対応できます。

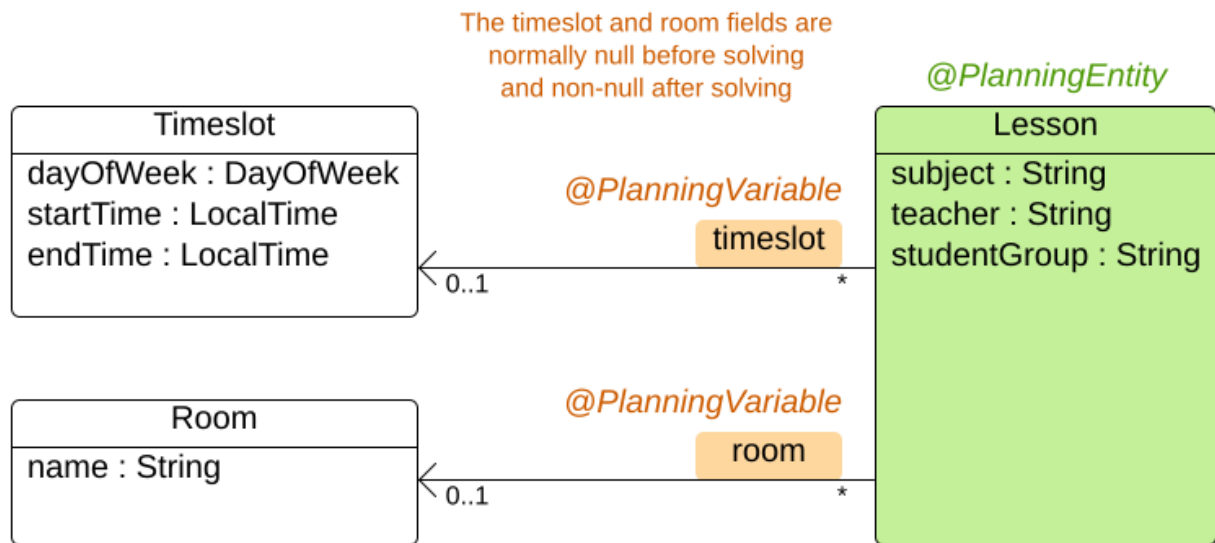
**Room** インスタンスは解決時に変化しないため、**Room** は **問題ファクト** でもあります。

### Lesson

授業中 (**Lesson** クラスで表現)、教師は複数の生徒に **Math by A.Turing for 9th grade**、**Chemistry by M.Curie for 10th grade** などの教科を指導します。ある教科について、毎週複数回、同じ教師が同じ生徒グループを指導する場合は、**Lesson** インスタンスが複数使用されますが、それらは **id** で識別可能です。たとえば、9 年生の場合は、1 週間に 6 回数学の授業があります。

解決中に、OptaPlanner は、**Lesson** クラスの **timeslot** フィールドと **room** フィールドを変更して、各授業を、時間枠 1 つ、部屋 1 つに割り当てます。OptaPlanner はこれらのフィールドを変更するため、**Lesson** は **プランニングエンティティ** となります。

## Time table class diagram



前図では、オレンジのフィールド以外のほぼすべてのフィールドに、入力データが含まれています。授業の **timeslot** フィールドと **room** フィールドは、入力データに割り当てられておらず (**null**)、出力データに割り当てられて (**null** ではない) います。Red Hat ビルドの OptaPlanner は、解決時にこれらのフィールドを変更します。このようなフィールドはプランニング変数と呼ばれます。このフィールドを OptaPlanner に認識させるには、**timeslot** フィールドと **room** のフィールドに **@PlanningVariable** アノテーションが必要です。このフィールドに含まれる **Lesson** クラスには、**@PlanningEntity** アノテーションが必要です。

### 手順

1. **src/main/java/com/example/domain/Timeslot.java** クラスを作成します。

```

package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
    public String toString() {
  
```

```

        return dayOfWeek + " " + startTime.toString();
    }

    // *****
    // Getters and setters
    // *****

    public DayOfWeek getDayOfWeek() {
        return dayOfWeek;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getEndTime() {
        return endTime;
    }
}

```

後述しているように、**toString()** メソッドで出力を短くするため、OptaPlanner の **DEBUG** ログまたは **TRACE** ログの読み取りが簡単になっています。

2. **src/main/java/com/example/domain/Room.java** クラスを作成します。

```

package com.example.domain;

public class Room {

    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // *****
    // Getters and setters
    // *****

    public String getName() {
        return name;
    }
}

```

3. **src/main/java/com/example/domain/Lesson.java** クラスを作成します。



```
package com.example.domain;

import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // *****
    // Getters and setters
    // *****

    public Long getId() {
        return id;
    }

    public String getSubject() {
        return subject;
    }

    public String getTeacher() {
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
```

```

        return timeslot;
    }

    public void setTimeslot(Timeslot timeslot) {
        this.timeslot = timeslot;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }
}

```

**Lesson** クラスには **@PlanningEntity** アノテーションが含まれており、その中にプランニング変数が1つ以上含まれているため、OptaPlanner はこのクラスが解決時に変化することを認識します。

**timeslot** フィールドには **@PlanningVariable** アノテーションがあるため、OptaPlanner は、このフィールドの値が変化することを認識しています。このフィールドに割り当てることのできる **Timeslot** インスタンスを見つけ出すために、OptaPlanner は **valueRangeProviderRefs** プロパティを使用して値の範囲プロバイダーと連携し、**List<Timeslot>** を提供して選択できるようにします。値の範囲プロバイダーに関する詳細は、「[プランニングソリューションでのドメインオブジェクトの収集](#)」を参照してください。

**room** フィールドにも、同じ理由で **@PlanningVariable** アノテーションが含まれます。

## 12.3. 制約の定義およびスコアの計算

問題の解決時に **スコア** で導かれた解の質を表します。スコアが高いほど質が高くなります。Red Hat ビルドの OptaPlanner は、利用可能な時間内で見つかった解の中から最高スコアのものを探し出します。これが **最適** 解である可能性があります。

時間割の例のユースケースでは、ハードとソフト制約を使用しているため、**HardSoftScore** クラスでスコアを表します。

- ハード制約は、絶対に違反しないでください。たとえば、**部屋に同時に割り当てることができ**  
**る授業は、最大1コマです。**
- ソフト制約は、違反しないようにしてください。たとえば、**教師は、1つの部屋での授業を希望**  
**します。**

ハード制約は、他のハード制約と比べて、重み付けを行います。ソフト制約は、他のソフト制約と比べて、重み付けを行います。ハード制約は、それぞれの重みに関係なく、常にソフト制約よりも高くなります。

**EasyScoreCalculator** クラスを実装して、スコアを計算できます。

```

public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

    @Override
    public HardSoftScore calculateScore(TimeTable timeTable) {
        List<Lesson> lessonList = timeTable.getLessonList();
    }
}

```

```

int hardScore = 0;
for (Lesson a : lessonList) {
    for (Lesson b : lessonList) {
        if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
            && a.getId() < b.getId()) {
            // A room can accommodate at most one lesson at the same time.
            if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                hardScore--;
            }
            // A teacher can teach at most one lesson at the same time.
            if (a.getTeacher().equals(b.getTeacher())) {
                hardScore--;
            }
            // A student can attend at most one lesson at the same time.
            if (a.getStudentGroup().equals(b.getStudentGroup())) {
                hardScore--;
            }
        }
    }
}
int softScore = 0;
// Soft constraints are only implemented in the "complete" implementation
return HardSoftScore.of(hardScore, softScore);
}
}

```

残念ながら、この解は漸増的ではないので、適切にスケーリングされません。授業が別の時間枠や教室に割り当てられるたびに、全授業が再評価され、新しいスコアが計算されます。

**src/main/java/com/example/solver/TimeTableConstraintProvider.java** クラスを作成して、漸増的スコア計算を実行すると、解がより優れたものになります。このクラスは、Java 8 Streams と SQL を基にした OptaPlanner の ConstraintStream API を使用します。**ConstraintProvider** は、**EasyScoreCalculator** と比べ、スケーリングの規模が遥かに大きくなっています ( $O(n^2)$  ではなく  $O(n)$ )。

## 手順

以下の **src/main/java/com/example/solver/TimeTableConstraintProvider.java** クラスを作成します。

```

package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
            // Hard constraints
            roomConflict(constraintFactory),

```

```

        teacherConflict(constraintFactory),
        studentGroupConflict(constraintFactory),
        // Soft constraints are only implemented in the "complete" implementation
    };
}

private Constraint roomConflict(ConstraintFactory constraintFactory) {
    // A room can accommodate at most one lesson at the same time.

    // Select a lesson ...
    return constraintFactory.from(Lesson.class)
        // ... and pair it with another lesson ...
        .join(Lesson.class,
            // ... in the same timeslot ...
            Joiners.equal(Lesson::getTimeslot),
            // ... in the same room ...
            Joiners.equal(Lesson::getRoom),
            // ... and the pair is unique (different id, no reverse pairs)
            Joiners.lessThan(Lesson::getId))
        // then penalize each pair with a hard weight.
        .penalize("Room conflict", HardSoftScore.ONE_HARD);
}

private Constraint teacherConflict(ConstraintFactory constraintFactory) {
    // A teacher can teach at most one lesson at the same time.
    return constraintFactory.from(Lesson.class)
        .join(Lesson.class,
            Joiners.equal(Lesson::getTimeslot),
            Joiners.equal(Lesson::getTeacher),
            Joiners.lessThan(Lesson::getId))
        .penalize("Teacher conflict", HardSoftScore.ONE_HARD);
}

private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
    // A student can attend at most one lesson at the same time.
    return constraintFactory.from(Lesson.class)
        .join(Lesson.class,
            Joiners.equal(Lesson::getTimeslot),
            Joiners.equal(Lesson::getStudentGroup),
            Joiners.lessThan(Lesson::getId))
        .penalize("Student group conflict", HardSoftScore.ONE_HARD);
}
}

```

## 12.4. プランニングソリューションでのドメインオブジェクトの収集

**TimeTable** インスタンスは、単一データセットの **Timeslot** インスタンス、**Room** インスタンス、および **Lesson** インスタンスをラップします。さらに、このインスタンスは、特定のプランニング変数の状態を持つ授業がすべて含まれているため、このインスタンスは **プランニングソリューション** となり、スコアが割り当てられます。

- 授業がまだ割り当てられていない場合は、スコアが **-4init/0hard/0soft** のソリューションなど、初期化されていないソリューションとなります。

- ハード制約に違反する場合、スコアが **-2hard/-3soft** のソリューションなど、**実行不可** なソリューションとなります。
- 全ハード制約に準拠している場合は、スコアが **0hard/-7soft** など、**実行可能** なソリューションとなります。

**TimeTable** クラスには **@PlanningSolution** アノテーションが含まれているため、Red Hat ビルドの OptaPlanner はこのクラスに全入出力データが含まれていることを認識します。

具体的には、このクラスは問題の入力です。

- 全時間枠が含まれる **timeslotList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全部屋が含まれる **roomList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全授業が含まれる **lessonList** フィールド
  - これは、解決時に変更されるため、プランニングエンティティです。
  - 各 **Lesson**:
    - **timeslot** フィールドおよび **room** フィールドの値は通常、**null** で未割り当てです。これらの値は、プランニング変数です。
    - **subject**、**teacher**、**studentGroup** などの他のフィールドは入力されます。これらのフィールドは問題プロパティです。

ただし、このクラスはソリューションの出力でもあります。

- **Lesson** インスタンスごとの **lessonList** フィールドには、解決後は **null** ではない **timeslot** フィールドと **room** フィールドが含まれます。
- 出力ソリューションの品質を表す **score** フィールド (例: **0hard/-5soft**)

## 手順

**src/main/java/com/example/domain/TimeTable.java** クラスを作成します。

```
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {

    @ValueRangeProvider(id = "timeslotRange")
    @ProblemFactCollectionProperty
```

```

private List<Timeslot> timeslotList;

@ValueRangeProvider(id = "roomRange")
@ProblemFactCollectionProperty
private List<Room> roomList;

@PlanningEntityCollectionProperty
private List<Lesson> lessonList;

@PlanningScore
private HardSoftScore score;

private TimeTable() {
}

public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
    List<Lesson> lessonList) {
    this.timeslotList = timeslotList;
    this.roomList = roomList;
    this.lessonList = lessonList;
}

// *****
// Getters and setters
// *****

public List<Timeslot> getTimeslotList() {
    return timeslotList;
}

public List<Room> getRoomList() {
    return roomList;
}

public List<Lesson> getLessonList() {
    return lessonList;
}

public HardSoftScore getScore() {
    return score;
}
}

```

## 値の範囲のプロバイダー

**timeslotList** フィールドは、値の範囲プロバイダーです。これは **Timeslot** インスタンスを保持し、OptaPlanner がこのインスタンスを選択して、**Lesson** インスタンスの **timeslot** フィールドに割り当てることができます。**timeslotList** フィールドには **@ValueRangeProvider** アノテーションがあり、**id** を、**Lesson** の **@PlanningVariable** の **valueRangeProviderRefs** に一致させます。

同じロジックに従い、**roomList** フィールドにも **@ValueRangeProvider** アノテーションが含まれています。

## 問題ファクトとプランニングエンティティのプロパティー

さらに OptaPlanner は、変更可能な **Lesson** インスタンス、さらに **TimeTableConstraintProvider** によるスコア計算に使用する **Timeslot** インスタンスと **Room** インスタンスを取得する方法を把握しておく必要があります。

**timeslotList** フィールドと **roomList** フィールドには **@ProblemFactCollectionProperty** アノテーションが含まれているため、**TimeTableConstraintProvider** はこれらのインスタンスから選択できます。

**lessonList** には **@PlanningEntityCollectionProperty** アノテーションが含まれているため、OptaPlanner は解決時に変更でき、**TimeTableConstraintProvider** はこの中から選択できます。

## 12.5. TIMETABLE サービスの作成

これですべて組み合わせ、REST サービスを作成する準備ができました。しかし、REST スレッドで計画問題を解決すると、HTTP タイムアウトの問題が発生します。そのため、Spring Boot スターターでは **SolverManager** を注入することで、個別のスレッドプールでソルバーを実行して複数のデータセットを並行して解決できます。

### 手順

**src/main/java/com/example/solver/TimeTableController.java** クラスを作成します。

```
package com.example.solver;

import java.util.UUID;
import java.util.concurrent.ExecutionException;

import com.example.domain.TimeTable;
import org.optaplanner.core.api.solver.SolverJob;
import org.optaplanner.core.api.solver.SolverManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/timeTable")
public class TimeTableController {

    @Autowired
    private SolverManager<TimeTable, UUID> solverManager;

    @PostMapping("/solve")
    public TimeTable solve(@RequestBody TimeTable problem) {
        UUID problemId = UUID.randomUUID();
        // Submit the problem to start solving
        SolverJob<TimeTable, UUID> solverJob = solverManager.solve(problemId, problem);
        TimeTable solution;
        try {
            // Wait until the solving ends
            solution = solverJob.getFinalBestSolution();
        } catch (InterruptedException | ExecutionException e) {
            throw new IllegalStateException("Solving failed.", e);
        }
        return solution;
    }
}
```

```
}
```

```
}
```

この例では、初期実装はソルバーが完了するのを待つので、HTTP タイムアウトがまだ発生します。**complete** 実装を使用することで、より適切に HTTP タイムアウトを回避できます。

## 12.6. ソルバー終了時間の設定

プランニングアプリケーションに終了設定または終了イベントがない場合、理論的には永続的に実行されることになり、実際には HTTP タイムアウトエラーが発生します。これが発生しないようにするには、**optaplanner.solver.termination.spent-limit** パラメーターを使用して、アプリケーションが終了してから時間を指定します。多くのアプリケーションでは、この時間を最低でも 5 分 (**5m**) に設定します。ただし、時間割の例では、解決時間を 5 分に制限すると、期間が十分に短いため HTTP タイムアウトを回避できます。

### 手順

以下の内容を含む **src/main/resources/application.properties** ファイルを作成します。

```
quarkus.optaplanner.solver.termination.spent-limit=5s
```

## 12.7. アプリケーションを実行可能にする手順

Red Hat ビルドの OptaPlanner Spring Boot の時間割プロジェクトを完了すると、標準 Java **main()** メソッドで駆動する 1 つの実行可能 JAR ファイルにすべてをパッケージ化します。

### 前提条件

- これで OptaPlanner Spring Boot の時間割プロジェクトが完成しました。

### 手順

1. 以下の内容を含む **TimeTableSpringBootApplication.java** クラスを作成します。

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TimeTableSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(TimeTableSpringBootApplication.class, args);
    }

}
```

2. Spring Initializr で作成された **src/main/java/com/example/DemoApplication.java** クラスは **TimeTableSpringBootApplication.java** クラスに置き換えます。
3. 通常の Java アプリケーションのメインクラスとして **TimeTableSpringBootApplication.java** クラスを実行します。



### 12.7.1. 時間割アプリケーションの試行

Red Hat ビルドの OptaPlanner Spring Boot 時間割アプリケーションの起動後に、任意の REST クライアントで REST サービスをテストできます。この例では Linux **curl** コマンドを使用して POST 要求を送信します。

#### 前提条件

- OptaPlanner Spring Boot アプリケーションが実行中である。

#### 手順

以下のコマンドを入力します。

```
$ curl -i -X POST http://localhost:8080/timeTable/solve -H "Content-Type:application/json" -d
'{"timeslotList":[{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"}],roomList":[{"name":"Room
A"}, {"name":"Room B"}],lessonList":[{"id":1,"subject":"Math","teacher":"A. Turing","studentGroup":"9th
grade"}, {"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th grade"},
{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th grade"},
{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th grade"}]}'
```

約 5 秒後 (**application.properties** で定義された終了時間) に、サービスにより、以下の例のような出力が返されます。

```
HTTP/1.1 200
Content-Type: application/json
...
```

```
{"timeslotList":..., "roomList":..., "lessonList":[{"id":1, "subject": "Math", "teacher": "A.
Turing", "studentGroup": "9th grade", "timeslot":
{"dayOfWeek": "MONDAY", "startTime": "08:30:00", "endTime": "09:30:00", "room": {"name": "Room A"}},
{"id":2, "subject": "Chemistry", "teacher": "M. Curie", "studentGroup": "9th grade", "timeslot":
{"dayOfWeek": "MONDAY", "startTime": "09:30:00", "endTime": "10:30:00", "room": {"name": "Room A"}},
{"id":3, "subject": "French", "teacher": "M. Curie", "studentGroup": "10th grade", "timeslot":
{"dayOfWeek": "MONDAY", "startTime": "08:30:00", "endTime": "09:30:00", "room": {"name": "Room B"}},
{"id":4, "subject": "History", "teacher": "I. Jones", "studentGroup": "10th grade", "timeslot":
{"dayOfWeek": "MONDAY", "startTime": "09:30:00", "endTime": "10:30:00", "room": {"name": "Room
B"}}], "score": "0hard/0soft"}
```

アプリケーションにより、4 つの授業がすべて 2 つの時間枠、そして 2 つある部屋のうちの 1 つに割り当てられている点に注目してください。また、すべてのハード制約に準拠することに注意してください。たとえば、M. Curie の 2 つの授業は異なる時間スロットにあります。

サーバー側で **info** ログに、この 5 秒間で OptaPlanner が行った内容が記録されます。

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score calculation
speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score calculation
speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed (28524/sec),
phase total (2), environment mode (REPRODUCIBLE).
```

## 12.7.2. アプリケーションのテスト

適切なアプリケーションにはテストが含まれます。以下の例では、Red Hat ビルドの OptaPlanner Spring Boot 時間割アプリケーションをテストします。このアプリケーションは、JUnit テストを使用してテストのデータセットを生成し、**TimeTableController** に送信して解決します。

### 手順

以下の内容を含む **src/test/java/com/example/solver/TimeTableControllerTest.java** クラスを作成します。

```
package com.example.solver;

import java.time.DayOfWeek;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

import com.example.domain.Lesson;
import com.example.domain.Room;
import com.example.domain.TimeTable;
import com.example.domain.Timeslot;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest(properties = {
    "optaplanner.solver.termination.spent-limit=1h", // Effectively disable this termination in favor of
    the best-score-limit
    "optaplanner.solver.termination.best-score-limit=0hard/*soft"})
public class TimeTableControllerTest {

    @Autowired
    private TimeTableController timeTableController;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableController.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(solution.getScore().isFeasible());
    }

    private TimeTable generateProblem() {
        List<Timeslot> timeslotList = new ArrayList<>();
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalDateTime.of(8, 30), LocalDateTime.of(9,
```

```

30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30), LocalTime.of(10,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30), LocalTime.of(11,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30), LocalTime.of(14,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30), LocalTime.of(15,
30)));

    List<Room> roomList = new ArrayList<>();
    roomList.add(new Room("Room A"));
    roomList.add(new Room("Room B"));
    roomList.add(new Room("Room C"));

    List<Lesson> lessonList = new ArrayList<>();
    lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
    lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
    lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
    lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
    lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

    lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
    lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
    lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
    lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
    lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
    return new TimeTable(timeslotList, roomList, lessonList);
}
}

```

このテストは、解決後にすべての授業がタイムスロットと部屋に割り当てられていることを確認します。また、実行可能解（ハード制約の違反なし）も確認します。

通常、ソルバーは 200 ミリ秒未満で実行可能解を検索します。**@SpringBootTest** アノテーションの **properties** が、実行可能なソリューション (**0hard/\*soft**) が見つかると同時に、ソルバーの終了を上書きします。こうすることで、ユニットテストが任意のハードウェアで実行される可能性があるため、ソルバーの時間をハードコード化するのを回避します。このアプローチを使用することで、動きが遅いシステムであっても、実行可能なソリューションを検索するのに十分な時間だけテストが実行されます。ただし、高速マシンでも、厳密に必要とされる時間よりもミリ秒単位で長く実行されることはありません。

### 12.7.3. ロギング

Red Hat ビルドの OptaPlanner Spring Boot の時間割アプリケーションアプリケーションを完了後にロギング情報を使用すると、**ConstraintProvider** で制約が微調整しやすくなります。**info** ログファイルでスコア計算の速度を確認して、制約に加えた変更の影響を評価します。デバッグモードでアプリケーションを実行して、アプリケーションが行う手順をすべて表示するか、追跡ログを使用して全手順および動きをロギングします。

#### 手順

1. 時間割アプリケーションを一定の時間 (例: 5 分) 実行します。
2. 以下の例のように、**log** ファイルのスコア計算の速度を確認します。

```
... Solving ended: ..., score calculation speed (29455/sec), ...
```

3. 制約を変更して、同じ時間、プランニングアプリケーションを実行し、**log** ファイルに記録されているスコア計算速度を確認します。
4. アプリケーションをデバッグモードで実行して、全手順をログに記録します。
  - コマンドラインからデバッグモードを実行するには、**-D** システムプロパティを使用します。
  - **application.properties** ファイルのロギングを変更するには、以下の行をこのファイルに追加します。

```
logging.level.org.optaplanner=debug
```

以下の例では、デバッグモードでの **log** ファイルの出力を表示します。

```
... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15),
picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
... CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15),
picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
...
```

5. **trace** ロギングを使用して、全手順、および手順ごとの全動きを表示します。

## 12.8. データベースと UI 統合の追加

Spring Boot を使用して Red Hat ビルドの OptaPlanner アプリケーションの例を作成したら、データベースと UI 統合を追加します。

### 前提条件/事前作業

- OptaPlanner Spring Boot の時間割サンプルが作成されている。

### 手順

1. **Timeslot**、**Room**、および **Lesson** の Java Persistence API (JPA) リポジトリを作成します。JPA リポジトリの作成に関する情報は、Spring の Web サイトの [Accessing Data with JPA](#) を参照してください。
2. REST で JPA リポジトリを公開します。リポジトリの公開に関する情報は、Spring の Web サイトの [Accessing JPA Data with REST](#) を参照してください。
3. **TimeTableRepository** Facade をビルドして、1回のトランザクションで **TimeTable** を読み取り、書き込みます。
4. 以下の例のように **TimeTableController** を調整します。

```
package com.example.solver;

import com.example.domain.TimeTable;
import com.example.persistence.TimeTableRepository;
```

```

import org.optaplanner.core.api.score.ScoreManager;
import org.optaplanner.core.api.solver.SolverManager;
import org.optaplanner.core.api.solver.SolverStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/timeTable")
public class TimeTableController {

    @Autowired
    private TimeTableRepository timeTableRepository;
    @Autowired
    private SolverManager<TimeTable, Long> solverManager;
    @Autowired
    private ScoreManager<TimeTable> scoreManager;

    // To try, GET http://localhost:8080/timeTable
    @GetMapping()
    public TimeTable getTimeTable() {
        // Get the solver status before loading the solution
        // to avoid the race condition that the solver terminates between them
        SolverStatus solverStatus = getSolverStatus();
        TimeTable solution =
timeTableRepository.findById(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
        scoreManager.updateScore(solution); // Sets the score
        solution.setSolverStatus(solverStatus);
        return solution;
    }

    @PostMapping("/solve")
    public void solve() {
        solverManager.solveAndListen(TimeTableRepository.SINGLETON_TIME_TABLE_ID,
            timeTableRepository::findById,
            timeTableRepository::save);
    }

    public SolverStatus getSolverStatus() {
        return
solverManager.getSolverStatus(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
    }

    @PostMapping("/stopSolving")
    public void stopSolving() {
        solverManager.terminateEarly(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
    }
}

```

簡素化するために、このコードは **TimeTable** インスタンスを1つだけ処理しますが、マルチテナントを有効にして、異なる高校の **TimeTable** インスタンスを複数、平行して処理することも簡単にできます。

**getTimeTable()** メソッドは、データベースから最新の時間割を返します。このメソッドは、**ScoreManager** (自動注入) を使用して、UI でスコアを表示できるように、対象の時間割のスコアを計算します。

**solve()** メソッドは、ジョブを開始して、現在の時間割を解決し、時間枠と部屋の割り当てをデータベースに保存します。このメソッドは、**SolverManager.solveAndListen()** メソッドを使用して、中間の最適解をリッスンし、それに合わせてデータベースを更新します。こうすることで、バックエンドで解決しながら、UI で進捗を表示できます。

5. **solve()** メソッドが即座に返されるようになったので、以下の例のように **TimeTableControllerTest** を調整します。

```
package com.example.solver;

import com.example.domain.Lesson;
import com.example.domain.TimeTable;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.optaplanner.core.api.solver.SolverStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest(properties = {
    "optaplanner.solver.termination.spent-limit=1h", // Effectively disable this termination in
    "optaplanner.solver.termination.best-score-limit=0hard/*soft"})
public class TimeTableControllerTest {

    @Autowired
    private TimeTableController timeTableController;

    @Test
    @Timeout(600_000)
    public void solveDemoDataUntilFeasible() throws InterruptedException {
        timeTableController.solve();
        TimeTable timeTable = timeTableController.getTimeTable();
        while (timeTable.getSolverStatus() != SolverStatus.NOT_SOLVING) {
            // Quick polling (not a Test Thread Sleep anti-pattern)
            // Test is still fast on fast systems and doesn't randomly fail on slow systems.
            Thread.sleep(20L);
            timeTable = timeTableController.getTimeTable();
        }
        assertFalse(timeTable.getLessonList().isEmpty());
        for (Lesson lesson : timeTable.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(timeTable.getScore().isFeasible());
    }
}
```

6. ソルバーの解決が完了するまで、最新のソリューションをポーリングします。
7. 時間割を視覚化するには、これらの REST メソッドの上に適切な Web UI を構築します。

## 12.9. MICROMETER と PROMETHEUS を使用して学校の時間割を監視する OPTAPLANNER SPRING BOOT アプリケーション

OptaPlanner は、Java アプリケーション用のメトリック計測ライブラリーである [Micrometer](#) を介してメトリックを公開します。Prometheus で Micrometer を使用して、学校の時間割アプリケーションで OptaPlanner ソルバーを監視できます。

### 前提条件

- Spring Boot OptaPlanner 学校の時間割アプリケーションを作成しました。
- Prometheus がインストールされている。Prometheus のインストールについては、[Prometheus](#) の Web サイトを参照してください。

### 手順

1. **technology/java-spring-boot** ディレクトリーに移動します。
2. 学校の時間割 **pom.xml** ファイルに Micrometer Prometheus 依存関係を追加します。

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

3. 次のプロパティを application.properties ファイルに追加します。

```
management.endpoints.web.exposure.include=metrics,prometheus
```

4. 学校の時間割アプリケーションを開始します。

```
mvn spring-boot:run
```

5. Web ブラウザーで <http://localhost:8080/actuator/prometheus> を開きます。



## 第13章 OPTAPLANNER と JAVA の RED HAT ビルド: 学校の時間割のクイックスタートガイド

本書では、OptaPlanner の制約解決人工知能 (AI) を使用してシンプル Java アプリケーションを作成するプロセスを説明します。生徒と教師のために学校の時間割を最適化するコマンドラインアプリケーションを作成します。

```
...
INFO Solving ended: time spent (5000), best score (0hard/9soft), ...
INFO
INFO |          | Room A   | Room B   | Room C   |
INFO |-----|-----|-----|-----|
INFO | MON 08:30 | English | Math    |          |
INFO |          | I. Jones | A. Turing |          |
INFO |          | 9th grade | 10th grade |          |
INFO |-----|-----|-----|-----|
INFO | MON 09:30 | History | Physics |          |
INFO |          | I. Jones | M. Curie |          |
INFO |          | 9th grade | 10th grade |          |
INFO |-----|-----|-----|-----|
INFO | MON 10:30 | History | Physics |          |
INFO |          | I. Jones | M. Curie |          |
INFO |          | 10th grade | 9th grade |          |
INFO |-----|-----|-----|-----|
...
INFO |-----|-----|-----|-----|
```

アプリケーションは、AI を使用してハードスケジュールとソフトスケジュールの **制約** を順守することにより、**Lesson** インスタンスを **タイムスロット** インスタンスと **Room** インスタンスに自動的に割り当てます。次に例を示します。

- 1 部屋に同時に割り当てることができる授業は、最大1コマです。
- 教師が同時に一度に行うことができる授業は最大1回です。
- 生徒は同時に出席できる授業は最大1コマです。
- 教師は、すべてのレッスンを同じ部屋で教えることを好みます。
- 教師は、連続した授業を好み、授業間に時間が空くの嫌います。
- 生徒は、同じ科目の連続したレッスンを嫌います。

数学的に考えると、学校の時間割は **NP 困難** の問題であります。これは、スケーリングが困難であることを意味します。すべての可能な組み合わせを単純に総当たりで反復すると、スーパーコンピュータ上でさえ、自明ではないデータセットに対して数百万年かかります。幸い、OptaPlanner などの AI 制約ソルバーには、妥当な時間内にほぼ最適なソリューションを提供する高度なアルゴリズムがあります。

### 前提条件

- OpenJDK (JDK) 11 がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスマーポータル (ログインが必要) の [ソフトウェアダウンロード](#) ページから入手できます。



- Apache Maven 3.6 以降がインストールされている。Maven は [Apache Maven Project](#) の Web サイトから入手できます。
- [IntelliJ IDEA](#)、VSCode、Eclipse などの IDE

## 13.1. MAVEN または GRADLE ビルドファイルを作成し、依存関係を追加する

OptaPlanner 学校の時間割アプリケーションには、Maven または Gradle を使用できます。ビルドファイルを作成したら、次の依存関係を追加します。

- 学校の時間割問題を解決するための **optaplanner-core** (コンパイルスコープ)
- **optaplanner-test** (テストスコープ) から JUnit への学校の時間割の制約のテスト
- OptaPlanner が実行するステップを表示するための **logback-classic** (ランタイムスコープ) などの実装

### 手順

1. Maven または Gradle ビルドファイルを作成します。
2. **optaplanner-core**、**optaplanner-test**、および **logback-classic** の依存関係をビルドファイルに追加します。
  - Maven の場合、次の依存関係を **pom.xml** ファイルに追加します。

```
<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-core</artifactId>
</dependency>

<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-test</artifactId>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.3</version>
</dependency>
```

次の例は、完全な **pom.xml** ファイルを示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.acme</groupId>
  <artifactId>optaplanner-hello-world-school-timetabling-quickstart</artifactId>
```

```

<version>1.0-SNAPSHOT</version>

<properties>
  <maven.compiler.release>11</maven.compiler.release>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  <version.org.optaplanner>8.11.1.Final-redhat-00006</version.org.optaplanner>
  <version.org.logback>1.2.3</version.org.logback>

  <version.compiler.plugin>3.8.1</version.compiler.plugin>
  <version.surefire.plugin>3.0.0-M5</version.surefire.plugin>
  <version.exec.plugin>3.0.0</version.exec.plugin>
</properties>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.optaplanner</groupId>
      <artifactId>optaplanner-bom</artifactId>
      <version>${version.org.optaplanner}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
      <version>${version.org.logback}</version>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-core</artifactId>
  </dependency>
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <scope>runtime</scope>
  </dependency>

  <!-- Testing -->
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${version.compiler.plugin}</version>
    </plugin>
  </plugins>
</build>

```

```

<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>${version.surefire.plugin}</version>
</plugin>
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <version>${version.exec.plugin}</version>
  <configuration>
    <mainClass>org.acme.schooltimetabling.TimeTableApp</mainClass>
  </configuration>
</plugin>
</plugins>
</build>

<repositories>
  <repository>
    <id>jboss-public-repository-group</id>
    <url>https://repository.jboss.org/nexus/content/groups/public/</url>
    <releases>
      <!-- Get releases only from Maven Central which is faster. -->
      <enabled>>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
</project>

```

- Gradle の場合、次の依存関係を **gradle.build** ファイルに追加します。

```

dependencies {
  implementation "org.optaplanner:optaplanner-core:${optaplannerVersion}"
  runtimeOnly "ch.qos.logback:logback-classic:${logbackVersion}"

  testImplementation "org.optaplanner:optaplanner-test:${optaplannerVersion}"
}

```

次の例は、完成した **gradle.build** ファイルを示しています。

```

plugins {
  id "java"
  id "application"
}

def optaplannerVersion = "{project-version}"
def logbackVersion = "1.2.3"

group = "org.acme"
version = "0.1.0-SNAPSHOT"

repositories {
  mavenCentral()
}

```

```
dependencies {
    implementation "org.optaplanner:optaplanner-core:${optaplannerVersion}"
    runtimeOnly "ch.qos.logback:logback-classic:${logbackVersion}"

    testImplementation "org.optaplanner:optaplanner-test:${optaplannerVersion}"
}

java {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}

compileJava {
    options.encoding = "UTF-8"
    options.compilerArgs << "-parameters"
}

compileTestJava {
    options.encoding = "UTF-8"
}

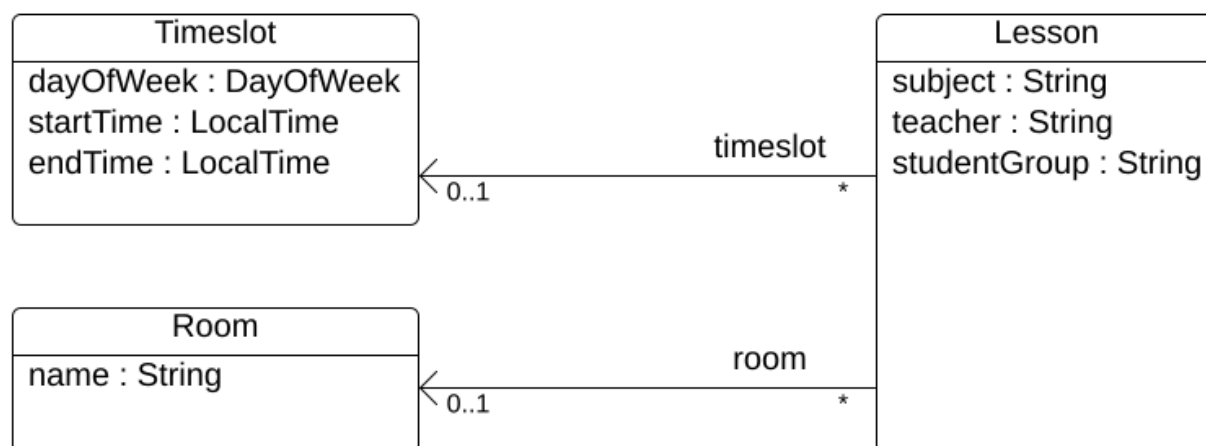
application {
    mainClass = "org.acme.schooltimetabling.TimeTableApp"
}

test {
    // Log the test execution results.
    testLogging {
        events "passed", "skipped", "failed"
    }
}
```

## 13.2. ドメインオブジェクトのモデル化

Red Hat ビルドの OptaPlanner の時間割プロジェクトの目標は、レッスンごとに時間枠と部屋に割り当てることです。これには、次の図に示すように、**Timeslot**、**Lesson**、および **Room** の3つのクラスを追加します。

## Time table class diagram



### Timeslot

**Timeslot** クラスは、**Monday 10:30 - 11:30**、**Tuesday 13:30 - 14:30** など、授業の長さを表します。この例では、時間枠はすべて同じ長さ (期間) で、昼休みまたは他の休憩時間にはこのスロットはありません。

高校のスケジュールは毎週 (同じ内容が) 繰り返されるだけなので、時間枠には日付がありません。また、[継続的プランニング](#) は必要ありません。解決時に **Timeslot** インスタンスが変更しないため、Timeslot は **問題ファクト** と呼ばれます。このようなクラスには OptaPlanner 固有のアノテーションは必要ありません。

### Room

**Room** クラスは、**Room A**、**Room B** など、授業の場所を表します。以下の例では、どの部屋も定員制限がなく、すべての授業に対応できます。

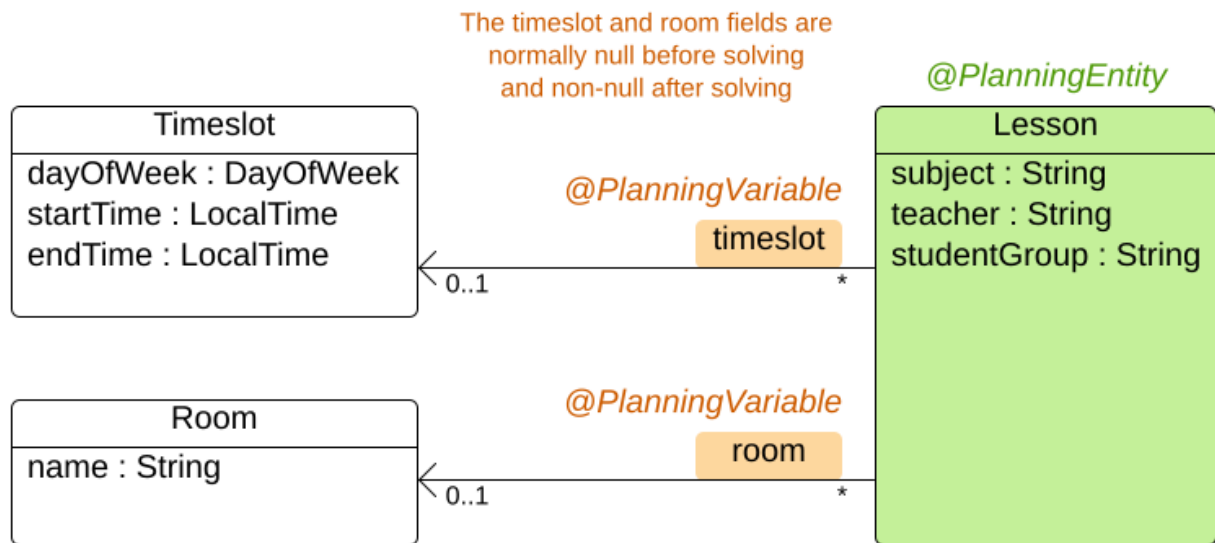
**Room** インスタンスは解決時に変化しないため、**Room** は **問題ファクト** でもあります。

### Lesson

授業中 (**Lesson** クラスで表現)、教師は複数の生徒に **Math by A.Turing for 9th grade**、**Chemistry by M.Curie for 10th grade** などの教科を指導します。ある教科について、毎週複数回、同じ教師が同じ生徒グループを指導する場合は、**Lesson** インスタンスが複数使用されますが、それらは **id** で識別可能です。たとえば、9 年生の場合は、1 週間に 6 回数学の授業があります。

解決中に、OptaPlanner は、**Lesson** クラスの **timeslot** フィールドと **room** フィールドを変更して、各授業を、時間枠 1 つ、部屋 1 つに割り当てます。OptaPlanner はこれらのフィールドを変更するため、**Lesson** は **プランニングエンティティ** となります。

## Time table class diagram



前図では、オレンジのフィールド以外のほぼすべてのフィールドに、入力データが含まれています。授業の **timeslot** フィールドと **room** フィールドは、入力データに割り当てられておらず (**null**)、出力データに割り当てられて (**null** ではない) います。Red Hat ビルドの OptaPlanner は、解決時にこれらのフィールドを変更します。このようなフィールドはプランニング変数と呼ばれます。このフィールドを OptaPlanner に認識させるには、**timeslot** フィールドと **room** のフィールドに **@PlanningVariable** アノテーションが必要です。このフィールドに含まれる **Lesson** クラスには、**@PlanningEntity** アノテーションが必要です。

### 手順

1. **src/main/java/com/example/domain/Timeslot.java** クラスを作成します。

```

package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
    public String toString() {
  
```

```

        return dayOfWeek + " " + startTime.toString();
    }

    // *****
    // Getters and setters
    // *****

    public DayOfWeek getDayOfWeek() {
        return dayOfWeek;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getEndTime() {
        return endTime;
    }
}

```

後述しているように、**toString()** メソッドで出力を短くするため、OptaPlanner の **DEBUG** ログまたは **TRACE** ログの読み取りが簡単になっています。

2. **src/main/java/com/example/domain/Room.java** クラスを作成します。

```

package com.example.domain;

public class Room {

    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // *****
    // Getters and setters
    // *****

    public String getName() {
        return name;
    }
}

```

3. **src/main/java/com/example/domain/Lesson.java** クラスを作成します。

```

package com.example.domain;

import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // *****
    // Getters and setters
    // *****

    public Long getId() {
        return id;
    }

    public String getSubject() {
        return subject;
    }

    public String getTeacher() {
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {

```



```

        return timeslot;
    }

    public void setTimeslot(Timeslot timeslot) {
        this.timeslot = timeslot;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }
}

```

**Lesson** クラスには **@PlanningEntity** アノテーションが含まれており、その中にプランニング変数が1つ以上含まれているため、OptaPlanner はこのクラスが解決時に変化することを認識します。

**timeslot** フィールドには **@PlanningVariable** アノテーションがあるため、OptaPlanner は、このフィールドの値が変化することを認識しています。このフィールドに割り当てることのできる **Timeslot** インスタンスを見つけ出すために、OptaPlanner は **valueRangeProviderRefs** プロパティを使用して値の範囲プロバイダーと連携し、**List<Timeslot>** を提供して選択できるようにします。値の範囲プロバイダーに関する詳細は、「[プランニングソリューションでのドメインオブジェクトの収集](#)」を参照してください。

**room** フィールドにも、同じ理由で **@PlanningVariable** アノテーションが含まれます。

### 13.3. 制約の定義およびスコアの計算

問題の解決時に **スコア** で導かれた解の質を表します。スコアが高いほど質が高くなります。Red Hat ビルドの OptaPlanner は、利用可能な時間内で見つかった解の中から最高スコアのものを探し出します。これが **最適** 解である可能性があります。

時間割の例のユースケースでは、ハードとソフト制約を使用しているため、**HardSoftScore** クラスでスコアを表します。

- ハード制約は、絶対に違反しないでください。たとえば、**部屋に同時に割り当てることができない授業は、最大1コマです。**
- ソフト制約は、違反しないようにしてください。たとえば、**教師は、1つの部屋での授業を希望します。**

ハード制約は、他のハード制約と比べて、重み付けを行います。ソフト制約は、他のソフト制約と比べて、重み付けを行います。ハード制約は、それぞれの重みに関係なく、常にソフト制約よりも高くなります。

**EasyScoreCalculator** クラスを実装して、スコアを計算できます。

```

public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

    @Override
    public HardSoftScore calculateScore(TimeTable timeTable) {
        List<Lesson> lessonList = timeTable.getLessonList();
    }
}

```

```

int hardScore = 0;
for (Lesson a : lessonList) {
    for (Lesson b : lessonList) {
        if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
            && a.getId() < b.getId()) {
            // A room can accommodate at most one lesson at the same time.
            if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                hardScore--;
            }
            // A teacher can teach at most one lesson at the same time.
            if (a.getTeacher().equals(b.getTeacher())) {
                hardScore--;
            }
            // A student can attend at most one lesson at the same time.
            if (a.getStudentGroup().equals(b.getStudentGroup())) {
                hardScore--;
            }
        }
    }
}
int softScore = 0;
// Soft constraints are only implemented in the "complete" implementation
return HardSoftScore.of(hardScore, softScore);
}
}

```

残念ながら、この解は漸増的ではないので、適切にスケーリングされません。授業が別の時間枠や教室に割り当てられるたびに、全授業が再評価され、新しいスコアが計算されます。

**src/main/java/com/example/solver/TimeTableConstraintProvider.java** クラスを作成して、漸増的スコア計算を実行すると、解がより優れたものになります。このクラスは、Java 8 Streams と SQL を基にした OptaPlanner の ConstraintStream API を使用します。**ConstraintProvider** は、**EasyScoreCalculator** と比べ、スケーリングの規模が遥かに大きくなっています ( $O(n^2)$  ではなく  $O(n)$ )。

## 手順

以下の **src/main/java/com/example/solver/TimeTableConstraintProvider.java** クラスを作成します。

```

package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
            // Hard constraints
            roomConflict(constraintFactory),

```

```

        teacherConflict(constraintFactory),
        studentGroupConflict(constraintFactory),
        // Soft constraints are only implemented in the "complete" implementation
    };
}

private Constraint roomConflict(ConstraintFactory constraintFactory) {
    // A room can accommodate at most one lesson at the same time.

    // Select a lesson ...
    return constraintFactory.from(Lesson.class)
        // ... and pair it with another lesson ...
        .join(Lesson.class,
            // ... in the same timeslot ...
            Joiners.equal(Lesson::getTimeslot),
            // ... in the same room ...
            Joiners.equal(Lesson::getRoom),
            // ... and the pair is unique (different id, no reverse pairs)
            Joiners.lessThan(Lesson::getId))
        // then penalize each pair with a hard weight.
        .penalize("Room conflict", HardSoftScore.ONE_HARD);
}

private Constraint teacherConflict(ConstraintFactory constraintFactory) {
    // A teacher can teach at most one lesson at the same time.
    return constraintFactory.from(Lesson.class)
        .join(Lesson.class,
            Joiners.equal(Lesson::getTimeslot),
            Joiners.equal(Lesson::getTeacher),
            Joiners.lessThan(Lesson::getId))
        .penalize("Teacher conflict", HardSoftScore.ONE_HARD);
}

private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
    // A student can attend at most one lesson at the same time.
    return constraintFactory.from(Lesson.class)
        .join(Lesson.class,
            Joiners.equal(Lesson::getTimeslot),
            Joiners.equal(Lesson::getStudentGroup),
            Joiners.lessThan(Lesson::getId))
        .penalize("Student group conflict", HardSoftScore.ONE_HARD);
}
}

```

## 13.4. プランニングソリューションでのドメインオブジェクトの収集

**TimeTable** インスタンスは、単一データセットの **Timeslot** インスタンス、**Room** インスタンス、および **Lesson** インスタンスをラップします。さらに、このインスタンスは、特定のプランニング変数の状態を持つ授業がすべて含まれているため、このインスタンスは **プランニングソリューション** となり、スコアが割り当てられます。

- 授業がまだ割り当てられていない場合は、スコアが **-4init/0hard/0soft** のソリューションなど、**初期化されていないソリューション**となります。

- ハード制約に違反する場合、スコアが **-2hard/-3soft** のソリューションなど、**実行不可** なソリューションとなります。
- 全ハード制約に準拠している場合は、スコアが **0hard/-7soft** など、**実行可能** なソリューションとなります。

**TimeTable** クラスには **@PlanningSolution** アノテーションが含まれているため、Red Hat ビルドの OptaPlanner はこのクラスに全入出力データが含まれていることを認識します。

具体的には、このクラスは問題の入力です。

- 全時間枠が含まれる **timeslotList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全部屋が含まれる **roomList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全授業が含まれる **lessonList** フィールド
  - これは、解決時に変更されるため、プランニングエンティティです。
  - 各 **Lesson**:
    - **timeslot** フィールドおよび **room** フィールドの値は通常、**null** で未割り当てです。これらの値は、プランニング変数です。
    - **subject**、**teacher**、**studentGroup** などの他のフィールドは入力されます。これらのフィールドは問題プロパティです。

ただし、このクラスはソリューションの出力でもあります。

- **Lesson** インスタンスごとの **lessonList** フィールドには、解決後は **null** ではない **timeslot** フィールドと **room** フィールドが含まれます。
- 出力ソリューションの品質を表す **score** フィールド (例: **0hard/-5soft**)

## 手順

**src/main/java/com/example/domain/TimeTable.java** クラスを作成します。

```
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {

    @ValueRangeProvider(id = "timeslotRange")
    @ProblemFactCollectionProperty
```

```

private List<Timeslot> timeslotList;

@ValueRangeProvider(id = "roomRange")
@ProblemFactCollectionProperty
private List<Room> roomList;

@PlanningEntityCollectionProperty
private List<Lesson> lessonList;

@PlanningScore
private HardSoftScore score;

private TimeTable() {
}

public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
    List<Lesson> lessonList) {
    this.timeslotList = timeslotList;
    this.roomList = roomList;
    this.lessonList = lessonList;
}

// *****
// Getters and setters
// *****

public List<Timeslot> getTimeslotList() {
    return timeslotList;
}

public List<Room> getRoomList() {
    return roomList;
}

public List<Lesson> getLessonList() {
    return lessonList;
}

public HardSoftScore getScore() {
    return score;
}
}

```

### 値の範囲のプロバイダー

**timeslotList** フィールドは、値の範囲プロバイダーです。これは **Timeslot** インスタンスを保持し、OptaPlanner がこのインスタンスを選択して、**Lesson** インスタンスの **timeslot** フィールドに割り当てることができます。**timeslotList** フィールドには **@ValueRangeProvider** アノテーションがあり、**id** を、**Lesson** の **@PlanningVariable** の **valueRangeProviderRefs** に一致させます。

同じロジックに従い、**roomList** フィールドにも **@ValueRangeProvider** アノテーションが含まれています。

### 問題ファクトとプランニングエンティティのプロパティー

さらに OptaPlanner は、変更可能な **Lesson** インスタンス、さらに **TimeTableConstraintProvider** によるスコア計算に使用する **Timeslot** インスタンスと **Room** インスタンスを取得する方法を把握しておく必要があります。

**timeslotList** フィールドと **roomList** フィールドには **@ProblemFactCollectionProperty** アノテーションが含まれているため、**TimeTableConstraintProvider** はこれらのインスタンスから選択できます。

**lessonList** には **@PlanningEntityCollectionProperty** アノテーションが含まれているため、OptaPlanner は解決時に変更でき、**TimeTableConstraintProvider** はこの中から選択できます。

## 13.5. TIMETABLEAPP.JAVA クラス

学校の時間割アプリケーションのすべてのコンポーネントを作成したら、それらをすべて **TimeTableApp.java** クラスにまとめます。

**main()** メソッドは以下のタスクを実行します。

1. **SolverFactory** を作成して、各データセットの **Solver** を構築します。
2. データセットをロードします。
3. **Solver.solve()** で解決します。
4. そのデータセットの解を視覚化します。

通常、アプリケーションには、解決する問題データセットごとに新しい **Solver** インスタンスを構築するための **SolverFactory** が1つあります。**SolverFactory** はスレッドセーフですが、**Solver** はそうではありません。学校の時間割アプリケーションの場合、データセットは1つしかないため、**Solver** インスタンスは1つだけです。

完成した **TimeTableApp.java** クラスは次のとおりです。

```
package org.acme.schooltimetabling;

import java.time.DayOfWeek;
import java.time.Duration;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import org.acme.schooltimetabling.domain.Lesson;
import org.acme.schooltimetabling.domain.Room;
import org.acme.schooltimetabling.domain.TimeTable;
import org.acme.schooltimetabling.domain.Timeslot;
import org.acme.schooltimetabling.solver.TimeTableConstraintProvider;
import org.optaplanner.core.api.solver.Solver;
import org.optaplanner.core.api.solver.SolverFactory;
import org.optaplanner.core.config.solver.SolverConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class TimeTableApp {
```

```

private static final Logger LOGGER = LoggerFactory.getLogger(TimeTableApp.class);

public static void main(String[] args) {
    SolverFactory<TimeTable> solverFactory = SolverFactory.create(new SolverConfig()
        .withSolutionClass(TimeTable.class)
        .withEntityClasses(Lesson.class)
        .withConstraintProviderClass(TimeTableConstraintProvider.class)
        // The solver runs only for 5 seconds on this small data set.
        // It's recommended to run for at least 5 minutes ("5m") otherwise.
        .withTerminationSpentLimit(Duration.ofSeconds(10)));

    // Load the problem
    TimeTable problem = generateDemoData();

    // Solve the problem
    Solver<TimeTable> solver = solverFactory.buildSolver();
    TimeTable solution = solver.solve(problem);

    // Visualize the solution
    printTimetable(solution);
}

public static TimeTable generateDemoData() {
    List<Timeslot> timeslotList = new ArrayList<>(10);
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30), LocalTime.of(9,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30), LocalTime.of(10,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30), LocalTime.of(11,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30), LocalTime.of(14,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30), LocalTime.of(15,
30)));

    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(8, 30), LocalTime.of(9,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(9, 30), LocalTime.of(10,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(10, 30), LocalTime.of(11,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(13, 30), LocalTime.of(14,
30)));
    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(14, 30), LocalTime.of(15,
30)));

    List<Room> roomList = new ArrayList<>(3);
    roomList.add(new Room("Room A"));
    roomList.add(new Room("Room B"));
    roomList.add(new Room("Room C"));

    List<Lesson> lessonList = new ArrayList<>();
    long id = 0;
    lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
    lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
    lessonList.add(new Lesson(id++, "Physics", "M. Curie", "9th grade"));
}

```



```

lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "9th grade"));
lessonList.add(new Lesson(id++, "Biology", "C. Darwin", "9th grade"));
lessonList.add(new Lesson(id++, "History", "I. Jones", "9th grade"));
lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));
lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));

lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
lessonList.add(new Lesson(id++, "Physics", "M. Curie", "10th grade"));
lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "10th grade"));
lessonList.add(new Lesson(id++, "French", "M. Curie", "10th grade"));
lessonList.add(new Lesson(id++, "Geography", "C. Darwin", "10th grade"));
lessonList.add(new Lesson(id++, "History", "I. Jones", "10th grade"));
lessonList.add(new Lesson(id++, "English", "P. Cruz", "10th grade"));
lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "10th grade"));

return new TimeTable(timeslotList, roomList, lessonList);
}

private static void printTimetable(TimeTable timeTable) {
    LOGGER.info("");
    List<Room> roomList = timeTable.getRoomList();
    List<Lesson> lessonList = timeTable.getLessonList();
    Map<Timeslot, Map<Room, List<Lesson>>> lessonMap = lessonList.stream()
        .filter(lesson -> lesson.getTimeslot() != null && lesson.getRoom() != null)
        .collect(Collectors.groupingBy(Lesson::getTimeslot,
Collectors.groupingBy(Lesson::getRoom)));
    LOGGER.info("|          | " + roomList.stream()
        .map(room -> String.format("%-10s", room.getName())).collect(Collectors.joining(" | ")) + "
|");
    LOGGER.info("|" + "-----|" .repeat(roomList.size() + 1));
    for (Timeslot timeslot : timeTable.getTimeslotList()) {
        List<List<Lesson>> cellList = roomList.stream()
            .map(room -> {
                Map<Room, List<Lesson>> byRoomMap = lessonMap.get(timeslot);
                if (byRoomMap == null) {
                    return Collections.<Lesson>emptyList();
                }
                List<Lesson> cellLessonList = byRoomMap.get(room);
                if (cellLessonList == null) {
                    return Collections.<Lesson>emptyList();
                }
                return cellLessonList;
            })
            .collect(Collectors.toList());

        LOGGER.info("| " + String.format("%-10s",
            timeslot.getDayOfWeek().toString().substring(0, 3) + " " + timeslot.getStartTime()) + " | "
            + cellList.stream().map(cellLessonList -> String.format("%-10s",
                cellLessonList.stream().map(Lesson::getSubject).collect(Collectors.joining(", "))))
            .collect(Collectors.joining(" | "))
            + " |");
        LOGGER.info("|          | "

```



```

        + cellList.stream().map(cellLessonList -> String.format("%-10s",
            cellLessonList.stream().map(Lesson::getTeacher).collect(Collectors.joining(", "))))
            .collect(Collectors.joining(" | "))
        + " |");
    LOGGER.info("|          | ")
        + cellList.stream().map(cellLessonList -> String.format("%-10s",
            cellLessonList.stream().map(Lesson::getStudentGroup).collect(Collectors.joining(",
"))))
            .collect(Collectors.joining(" | "))
        + " |");
    LOGGER.info("|" + "-----|" .repeat(roomList.size() + 1));
}
List<Lesson> unassignedLessons = lessonList.stream()
    .filter(lesson -> lesson.getTimeslot() == null || lesson.getRoom() == null)
    .collect(Collectors.toList());
if (!unassignedLessons.isEmpty()) {
    LOGGER.info("");
    LOGGER.info("Unassigned lessons");
    for (Lesson lesson : unassignedLessons) {
        LOGGER.info(" " + lesson.getSubject() + " - " + lesson.getTeacher() + " - " +
lesson.getStudentGroup());
    }
}
}
}
}

```

**main ()** メソッドは最初に **SolverFactory** を作成します。

```

SolverFactory<TimeTable> solverFactory = SolverFactory.create(new SolverConfig()
    .withSolutionClass(TimeTable.class)
    .withEntityClasses(Lesson.class)
    .withConstraintProviderClass(TimeTableConstraintProvider.class)
    // The solver runs only for 5 seconds on this small data set.
    // It's recommended to run for at least 5 minutes ("5m") otherwise.
    .withTerminationSpentLimit(Duration.ofSeconds(5)));

```

**SolverFactory** の作成により、以前に作成した **@PlanningSolution** クラス、**@PlanningEntity** クラス、および **ConstraintProvider** クラスが登録されます。

終了設定または **terminationEarly()** イベントがない場合、ソルバーは永久に実行されます。これを避けるために、ソルバーは解決時間を 5 秒に制限します。

5 秒後、**main()** メソッドは問題をロードして解決し、解決策を出力します。

```

// Load the problem
TimeTable problem = generateDemoData();

// Solve the problem
Solver<TimeTable> solver = solverFactory.buildSolver();
TimeTable solution = solver.solve(problem);

// Visualize the solution
printTimetable(solution);

```

**solve()** メソッドはすぐには戻りません。最適解を返す前に 5 秒間実行されます。

OptaPlanner は、利用可能な終了時間内に見つかった**最適なソリューション**を返します。NP 困難な問題の性質上、特に大規模なデータセットの場合、最適解が最適ではない可能性があります。終了時間を増やして、より良い解決策を見つけてください。

**generateDemoData()** メソッドは、解決する学校の時間割の問題を生成します。

**printTimetable()** メソッドは時刻表をコンソールにきれいに出力するので、スケジュールが適切かどうかを視覚的に簡単に判断できます。

## 13.6. 学校の時間割アプリケーションを作成して実行する

学校の時間割 Java アプリケーションのすべてのコンポーネントが完成したので、それらをすべて **TimeTableApp.java** クラスにまとめて実行する準備が整いました。

### 前提条件

- 学校の時間割アプリケーションに必要なすべてのコンポーネントを作成しました。

### 手順

1. **src/main/java/org/acme/schooltimetabling/TimeTableApp.java** クラスを作成します。

```
package org.acme.schooltimetabling;

import java.time.DayOfWeek;
import java.time.Duration;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import org.acme.schooltimetabling.domain.Lesson;
import org.acme.schooltimetabling.domain.Room;
import org.acme.schooltimetabling.domain.TimeTable;
import org.acme.schooltimetabling.domain.Timeslot;
import org.acme.schooltimetabling.solver.TimeTableConstraintProvider;
import org.optaplanner.core.api.solver.Solver;
import org.optaplanner.core.api.solver.SolverFactory;
import org.optaplanner.core.config.solver.SolverConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class TimeTableApp {

    private static final Logger LOGGER = LoggerFactory.getLogger(TimeTableApp.class);

    public static void main(String[] args) {
        SolverFactory<TimeTable> solverFactory = SolverFactory.create(new SolverConfig()
            .withSolutionClass(TimeTable.class)
            .withEntityClasses(Lesson.class)
            .withConstraintProviderClass(TimeTableConstraintProvider.class)
```

```

        // The solver runs only for 5 seconds on this small data set.
        // It's recommended to run for at least 5 minutes ("5m") otherwise.
        .withTerminationSpentLimit(Duration.ofSeconds(10)));

    // Load the problem
    TimeTable problem = generateDemoData();

    // Solve the problem
    Solver<TimeTable> solver = solverFactory.buildSolver();
    TimeTable solution = solver.solve(problem);

    // Visualize the solution
    printTimetable(solution);
}

public static TimeTable generateDemoData() {
    List<Timeslot> timeslotList = new ArrayList<>(10);
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
    LocalTime.of(9, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
    LocalTime.of(10, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
    LocalTime.of(11, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
    LocalTime.of(14, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
    LocalTime.of(15, 30)));

    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(8, 30),
    LocalTime.of(9, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(9, 30),
    LocalTime.of(10, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(10, 30),
    LocalTime.of(11, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(13, 30),
    LocalTime.of(14, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(14, 30),
    LocalTime.of(15, 30)));

    List<Room> roomList = new ArrayList<>(3);
    roomList.add(new Room("Room A"));
    roomList.add(new Room("Room B"));
    roomList.add(new Room("Room C"));

    List<Lesson> lessonList = new ArrayList<>();
    long id = 0;
    lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
    lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
    lessonList.add(new Lesson(id++, "Physics", "M. Curie", "9th grade"));
    lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "9th grade"));
    lessonList.add(new Lesson(id++, "Biology", "C. Darwin", "9th grade"));
    lessonList.add(new Lesson(id++, "History", "I. Jones", "9th grade"));
    lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
    lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
    lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));
    lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));
}

```

```

lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
lessonList.add(new Lesson(id++, "Physics", "M. Curie", "10th grade"));
lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "10th grade"));
lessonList.add(new Lesson(id++, "French", "M. Curie", "10th grade"));
lessonList.add(new Lesson(id++, "Geography", "C. Darwin", "10th grade"));
lessonList.add(new Lesson(id++, "History", "I. Jones", "10th grade"));
lessonList.add(new Lesson(id++, "English", "P. Cruz", "10th grade"));
lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "10th grade"));

return new TimeTable(timeslotList, roomList, lessonList);
}

private static void printTimetable(TimeTable timeTable) {
    LOGGER.info("");
    List<Room> roomList = timeTable.getRoomList();
    List<Lesson> lessonList = timeTable.getLessonList();
    Map<Timeslot, Map<Room, List<Lesson>>> lessonMap = lessonList.stream()
        .filter(lesson -> lesson.getTimeslot() != null && lesson.getRoom() != null)
        .collect(Collectors.groupingBy(Lesson::getTimeslot,
Collectors.groupingBy(Lesson::getRoom)));
    LOGGER.info("|          | " + roomList.stream()
        .map(room -> String.format("%-10s", room.getName())).collect(Collectors.joining(" |
")) + " |");
    LOGGER.info("| " + "-----|".repeat(roomList.size() + 1));
    for (Timeslot timeslot : timeTable.getTimeslotList()) {
        List<List<Lesson>> cellList = roomList.stream()
            .map(room -> {
                Map<Room, List<Lesson>> byRoomMap = lessonMap.get(timeslot);
                if (byRoomMap == null) {
                    return Collections.<Lesson>emptyList();
                }
                List<Lesson> cellLessonList = byRoomMap.get(room);
                if (cellLessonList == null) {
                    return Collections.<Lesson>emptyList();
                }
                return cellLessonList;
            })
            .collect(Collectors.toList());

        LOGGER.info("| " + String.format("%-10s",
            timeslot.getDayOfWeek().toString().substring(0, 3) + " " +
timeslot.getStartTime()) + " | "
            + cellList.stream().map(cellLessonList -> String.format("%-10s",
cellLessonList.stream().map(Lesson::getSubject).collect(Collectors.joining(", "))))
                .collect(Collectors.joining(" | "))
            + " |");
        LOGGER.info("|          | "
            + cellList.stream().map(cellLessonList -> String.format("%-10s",
cellLessonList.stream().map(Lesson::getTeacher).collect(Collectors.joining(", "))))
                .collect(Collectors.joining(" | "))
            + " |");
    }
}

```

```

        LOGGER.info("|          | "
            + cellList.stream().map(cellLessonList -> String.format("%-10s",
cellLessonList.stream().map(Lesson::getStudentGroup).collect(Collectors.joining(", ")))
                .collect(Collectors.joining(" | ")))
            + " |");
        LOGGER.info("|" + "-----|" .repeat(roomList.size() + 1));
    }
    List<Lesson> unassignedLessons = lessonList.stream()
        .filter(lesson -> lesson.getTimeslot() == null || lesson.getRoom() == null)
        .collect(Collectors.toList());
    if (!unassignedLessons.isEmpty()) {
        LOGGER.info("");
        LOGGER.info("Unassigned lessons");
        for (Lesson lesson : unassignedLessons) {
            LOGGER.info(" " + lesson.getSubject() + " - " + lesson.getTeacher() + " - " +
lesson.getStudentGroup());
        }
    }
}
}
}

```

2. **TimeTableApp** クラスを通常の Java アプリケーションのメインクラスとして実行します。次の出力が得られるはずです。

```

...
INFO |          | Room A   | Room B   | Room C   |
INFO |-----|-----|-----|-----|
INFO | MON 08:30 | English | Math     |          |
INFO |          | I. Jones | A. Turing |          |
INFO |          | 9th grade | 10th grade |          |
INFO |-----|-----|-----|-----|
INFO | MON 09:30 | History | Physics  |          |
INFO |          | I. Jones | M. Curie  |          |
INFO |          | 9th grade | 10th grade |          |
...

```

3. コンソール出力を確認します。すべての厳しい制約に準拠していますか?**TimeTableConstraintProvider** の **roomConflict** 制約をコメントアウトするとどうなりますか?

**info** ログは、OptaPlanner がその 5 秒間に何をしたかを示しています。

```

... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score calculation
speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score calculation
speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed (28524/sec),
phase total (2), environment mode (REPRODUCIBLE).

```

## 13.7. アプリケーションのテスト

適切なアプリケーションにはテストが含まれます。timetable プロジェクトで制約とソルバーをテストします。

### 13.7.1. 学校の時間割の制約をテストする

timetable プロジェクトの各制約を個別にテストするには、単体テストで **ConstraintVerifier** を使用します。これにより、各制約のコーナーケースが他のテストから分離されてテストされるため、適切なテストカバレッジで新しい制約を追加する際のメンテナンスが軽減されます。

このテストは、制約 **TimeTableConstraintProvider::roomConflict** が、同じ部屋で3つのレッスンを与えられ、そのうちの2つのレッスンが同じタイムスロットを持つ場合、一致の重み1でペナルティを課すことを検証します。したがって、制約の重みが **10hard** の場合、スコアは **-10hard** 減少します。

#### 手順

**src/test/java/org/acme/optaplanner/solver/TimeTableConstraintProviderTest.java** クラスを作成します。

```
package org.acme.optaplanner.solver;

import java.time.DayOfWeek;
import java.time.LocalTime;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.Room;
import org.acme.optaplanner.domain.TimeTable;
import org.acme.optaplanner.domain.Timeslot;
import org.junit.jupiter.api.Test;
import org.optaplanner.test.api.score.stream.ConstraintVerifier;

@QuarkusTest
class TimeTableConstraintProviderTest {

    private static final Room ROOM = new Room("Room1");
    private static final Timeslot TIMESLOT1 = new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9,0), LocalTime.NOON);
    private static final Timeslot TIMESLOT2 = new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(9,0), LocalTime.NOON);

    @Inject
    ConstraintVerifier<TimeTableConstraintProvider, TimeTable> constraintVerifier;

    @Test
    void roomConflict() {
        Lesson firstLesson = new Lesson(1, "Subject1", "Teacher1", "Group1");
        Lesson conflictingLesson = new Lesson(2, "Subject2", "Teacher2", "Group2");
        Lesson nonConflictingLesson = new Lesson(3, "Subject3", "Teacher3", "Group3");

        firstLesson.setRoom(ROOM);
        firstLesson.setTimeslot(TIMESLOT1);

        conflictingLesson.setRoom(ROOM);
        conflictingLesson.setTimeslot(TIMESLOT1);
    }
}
```

```

        nonConflictingLesson.setRoom(ROOM);
        nonConflictingLesson.setTimeslot(TIMESLOT2);

        constraintVerifier.verifyThat(TimeTableConstraintProvider::roomConflict)
            .given(firstLesson, conflictingLesson, nonConflictingLesson)
            .penalizesBy(1);
    }
}

```

制約の重みが **ConstraintProvider** でハードコーディングされている場合でも、**ConstraintVerifier** がテスト中に制約の重みを無視することに注意してください。これは、実稼動に入る前に制約の重みが定期的に変更されるためです。このように、制約の重みの微調整によって単体テストが中断されることはありません。

### 13.7.2. 学校の時間割ソルバーをテストする

以下の例では、Red Hat ビルドの Quarkus で Red Hat ビルドの OptaPlanner の時間割プロジェクトをテストします。このアプリケーションは、JUnit テストを使用してテストのデータセットを生成し、**TimeTableController** に送信して解決します。

#### 手順

1. 以下の内容を含む **src/test/java/com/example/rest/TimeTableResourceTest.java** クラスを作成します。

```

package com.exmaple.optaplanner.rest;

import java.time.DayOfWeek;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import com.exmaple.optaplanner.domain.Room;
import com.exmaple.optaplanner.domain.Timeslot;
import com.exmaple.optaplanner.domain.Lesson;
import com.exmaple.optaplanner.domain.TimeTable;
import com.exmaple.optaplanner.rest.TimeTableResource;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

```



```

@Test
@Timeout(600_000)
public void solve() {
    TimeTable problem = generateProblem();
    TimeTable solution = timeTableResource.solve(problem);
    assertFalse(solution.getLessonList().isEmpty());
    for (Lesson lesson : solution.getLessonList()) {
        assertNotNull(lesson.getTimeslot());
        assertNotNull(lesson.getRoom());
    }
    assertTrue(solution.getScore().isFeasible());
}

private TimeTable generateProblem() {
    List<Timeslot> timeslotList = new ArrayList<>();
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
    LocalTime.of(9, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
    LocalTime.of(10, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
    LocalTime.of(11, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
    LocalTime.of(14, 30)));
    timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
    LocalTime.of(15, 30)));

    List<Room> roomList = new ArrayList<>();
    roomList.add(new Room("Room A"));
    roomList.add(new Room("Room B"));
    roomList.add(new Room("Room C"));

    List<Lesson> lessonList = new ArrayList<>();
    lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
    lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
    lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
    lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
    lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

    lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
    lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
    lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
    lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
    lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
    return new TimeTable(timeslotList, roomList, lessonList);
}
}

```

このテストは、解決後にすべての授業がタイムスロットと部屋に割り当てられていることを確認します。また、実行可能解（ハード制約の違反なし）も確認します。

2. テストプロパティを **src / main / resources /application.properties** ファイルに追加します。

```

# The solver runs only for 5 seconds to avoid a HTTP timeout in this simple implementation.
# It's recommended to run for at least 5 minutes ("5m") otherwise.

```



```
quarkus.optaplanner.solver.termination.spent-limit=5s

# Effectively disable this termination in favor of the best-score-limit
%test.quarkus.optaplanner.solver.termination.spent-limit=1h
%test.quarkus.optaplanner.solver.termination.best-score-limit=0hard/*soft
```

通常、ソルバーは 200 ミリ秒未満で実行可能解を検索します。**application.properties** が、実行可能なソリューション (**0hard/\*soft**) が見つかりと同時に終了するように、テスト中のソルバーの終了を上書きします。こうすることで、ユニットテストが任意のハードウェアで実行される可能性があるため、ソルバーの時間をハードコード化するのを回避します。このアプローチを使用することで、動きが遅いシステムであっても、実行可能なソリューションを検索するのに十分な時間だけテストが実行されます。ただし、高速システムでも、厳密に必要とされる時間よりもミリ秒単位で長く実行されることはありません。

## 13.8. ロギング

Red Hat ビルドの OptaPlanner の時間割プロジェクトを完了後にロギング情報を使用すると、**ConstraintProvider** で制約が微調整しやすくなります。**info** ログファイルでスコア計算の速度を確認して、制約に加えた変更の影響を評価します。デバッグモードでアプリケーションを実行して、アプリケーションが行う手順をすべて表示するか、追跡ログを使用して全手順および動きをロギングします。

### 手順

1. 時間割アプリケーションを一定の時間 (例: 5 分) 実行します。
2. 以下の例のように、**log** ファイルのスコア計算の速度を確認します。

```
... Solving ended: ..., score calculation speed (29455/sec), ...
```

3. 制約を変更して、同じ時間、プランニングアプリケーションを実行し、**log** ファイルに記録されているスコア計算速度を確認します。
4. アプリケーションをデバッグモードで実行して、アプリケーションの全実行ステップをログに記録します。
  - コマンドラインからデバッグモードを実行するには、**-D** システムプロパティを使用します。
  - デバッグモードを永続的に有効にするには、以下の行を **application.properties** ファイルに追加します。

```
quarkus.log.category."org.optaplanner".level=debug
```

以下の例では、デバッグモードでの **log** ファイルの出力を表示します。

```
... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15),
picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
... CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15),
picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
...
```

5. **trace** ロギングを使用して、全手順、および手順ごとの全動きを表示します。

## 13.9. MICROMETER と PROMETHEUS を使用して学校の時間割を監視する OPTAPLANNER JAVA アプリケーション

OptaPlanner は、Java アプリケーション用のメトリック計測ライブラリーである [Micrometer](#) を介してメトリックを公開します。Prometheus で Micrometer を使用して、学校の時間割アプリケーションで OptaPlanner ソルバーを監視できます。

### 前提条件

- OptaPlanner 学校の時間割アプリケーションを Java で作成しました。
- Prometheus がインストールされている。Prometheus のインストールについては、[Prometheus](#) の Web サイトを参照してください。

### 手順

1. Micrometer Prometheus 依存関係を学校の時間割 **pom.xml** ファイルに追加します。**<MICROMETER\_VERSION>** は、インストールした Micrometer のバージョンです。

```
<dependency>
<groupId>io.micrometer</groupId>
<artifactId>micrometer-registry-prometheus</artifactId>
<version><MICROMETER_VERSION></version>
</dependency>
```



#### 注記

**micrometer-core** 依存関係も必要です。ただし、この依存関係は **optaplanner-core** 依存関係に含まれているため、**pom.xml** ファイルに追加する必要はありません。

2. 次の import ステートメントを **TimeTableApp.java** クラスに追加します。

```
import io.micrometer.core.instrument.Metrics;
import io.micrometer.prometheus.PrometheusConfig;
import io.micrometer.prometheus.PrometheusMeterRegistry;
```

3. **TimeTableApp.java** クラスのメインメソッドの先頭に次の行を追加して、ソリューションが開始する前に Prometheus が **com.sun.net.httpserver.HttpServer** からデータを破棄できるようにします。

```
PrometheusMeterRegistry prometheusRegistry = new
PrometheusMeterRegistry(PrometheusConfig.DEFAULT);

try {
    HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
    server.createContext("/prometheus", httpExchange -> {
        String response = prometheusRegistry.scrape();
        httpExchange.sendResponseHeaders(200, response.getBytes().length);
        try (OutputStream os = httpExchange.getResponseBody()) {
            os.write(response.getBytes());
        }
    });
}
```

```

    }
  });

  new Thread(server::start).start();
} catch (IOException e) {
  throw new RuntimeException(e);
}

Metrics.addRegistry(prometheusRegistry);

solve();
}

```

4. 次の行を追加して、解決時間を制御します。解決時間を調整することで、解決に費やされた時間に基づいて指標がどのように変化するかを確認できます。

```
withTerminationSpentLimit(Duration.ofMinutes(5));
```

5. 学校の時間割アプリケーションを開始します。
6. Web ブラウザーで <http://localhost:8080/prometheus> を開き、Prometheus で timetable アプリケーションを表示します。
7. 監視システムを開いて、OptaPlanner プロジェクトのメトリックを表示します。  
次のメトリックが公開されます。

- **optaplanner\_solver\_errors\_total**: 測定開始以降に解決中に発生したエラーの総数。
- **optaplanner\_solver\_solve\_duration\_seconds\_active\_count**: 現在解決しているソルバーの数。
- **optaplanner\_solver\_solve\_duration\_seconds\_max**: 現在アクティブなソルバーの実行時間が最も長い実行時間。
- **optaplanner\_solver\_solve\_duration\_seconds\_duration\_sum**: アクティブな各ソルバーの解決時間の合計。たとえば、アクティブなソルバーが2つあり、一方が3分間実行され、もう一方が1分間実行されている場合、合計計算時間は4分です。

## パート V. RED HAT ビルドの OPTAPLANNER のスターターアプリケーション

Red Hat ビルドの OptaPlanner は、Red Hat OpenShift Container Platform でそのままデプロイできる以下のスターターアプリケーションを提供します。

- 従業員勤務表スターターアプリケーション
- 運搬経路計画スターターアプリケーション

OptaPlanner スターターアプリケーションは、サンプルおよびクイックスタートガイドよりも進んでいます。これらは特定のユースケースに重点が置かれ、最適な技術を使用して、プランニングソリューションを構築します。

## 第14章 IDE での RED HAT ビルドの OPTAPLANNER の使用: 従業員の勤務表サンプル

ビジネスルールの作成者は、IDE を使用して Red Hat ビルドの OptaPlanner 機能を使用する **optaweb-employee-rostering** スターターアプリケーションのビルド、実行、および変更が可能です。

### 前提条件

- Red Hat CodeReady Studio、IntelliJ IDEA などの統合開発環境を使用すること。
- Java 言語を理解していること。
- React および TypeScript を理解していること。この要件は、OptaWeb UI を開発するために必要です。

### 14.1. 従業員勤務表スターターアプリケーションの概要

従業員勤務表スターターアプリケーションは、組織内のさまざまな場所に従業員を割り当てます。たとえば、アプリケーションを使用して、病院での看護師のシフト、さまざまな場所での警備勤務シフト、作業者の組み立てラインのシフトを割り当てます。

従業員勤務表を最適化するには、多くの変数を考慮する必要があります。たとえば、業務が異なれば、求められるスキルが異なります。また、従業員の中には、特定の時間帯に勤務できない場合や、特定の時間帯での勤務を希望する場合があります。さらに、従業員によっては、1回に就業できる時間に制限がある契約を交わしている可能性があります。

このスターターアプリケーションの Red Hat ビルドの OptaPlanner ルールは、ハード制約およびソフト制約を使用します。最適化時に、従業員が勤務できない(または病欠の)場合や、ある1つのシフト内の2つのスポットで働くことができない場合など、プランニングエンジンはハード制約に違反することができません。プランニングエンジンは、ソフト制約(特定のシフトで勤務しないという従業員の希望など)に順守しようとしますが、最適なソリューションには違反が必要だと判断した場合は、違反することができます。

### 14.2. 従業員勤務表スターターアプリケーションの構築と実行

ソースコードから従業員勤務表スターターアプリケーションを構築して、JAR ファイルとして実行します。

または、Eclipse (Red Hat JBoss CodeReady Studio を含む) などの IDE を使用して、アプリケーションを構築し、実行します。

#### 14.2.1. デプロイメントファイルの準備

デプロイメントファイルをダウンロードし、準備してから、アプリケーションの構築、デプロイを行う必要があります。

### 手順

1. Red Hat カスタマーポータルの [Software Downloads](#) ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
  - Product: Decision Manager
  - Version: 7.12

2. Red Hat Decision Manager 7.12.0 Kogito および OptaPlanner 8 Decision Services クイックスタート (**rhdm-7.12.0-kogito-and-optaplanner-quickstarts.zip**) をダウンロードします。
3. **rhdm-7.12.0-kogito-and-optaplanner-quickstarts.zip** ファイルを展開します。
4. Red Hat Decision Manager 7.12 Maven リポジトリ Kogito および OptaPlanner 8 Maven リポジトリ (**rhdm-7.12.0-kogito-maven-repository.zip**) をダウンロードします。
5. **rhdm-7.12.0-kogito-maven-repository.zip** ファイルを抽出します。
6. **rhdm-7.12.0-kogito-maven-repository / maven-repository** サブディレクトリの内容を **~/ .m2 / repository** ディレクトリにコピーします。
7. **optaweb-8.11.1.Final-redhat-00006/optaweb-employee-rostering** ディレクトリに移動します。このディレクトリは、後述する例でベースとなるディレクトリです。



### 注記

ファイルおよびディレクトリの名前で使用されるバージョン番号は、本書で使用するバージョンよりも新しい場合があります。

## 14.2.2. 従業員勤務表スターターアプリケーションの JAR ファイル実行

Red Hat Decision Manager 7.12.0 Kogito および OptaPlanner 8 Decision Services クイックスタートのダウンロードに含まれている JAR ファイルから従業員名簿スターターアプリケーションを実行できます。

### 前提条件

- 「[デプロイメントファイルの準備](#)」の説明に従って、**rhpm-7.12.0-kogito-and-optaplanner-quickstarts.zip** ファイルをダウンロードして展開している。
- Java 開発キットがインストールされている。
- Maven がインストールされている。
- ホストからインターネットにアクセスできる。ビルドプロセスは、インターネットを使用して、外部のリポジトリから Maven パッケージをダウンロードします。

### 手順

1. コマンドターミナルで、**rhdm-7.12.0-kogito-and-optaplanner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-employee-rostering** ディレクトリに移動します。
2. 以下のコマンドを入力します。  

```
mvn clean install -DskipTests
```
3. ビルドプロセスが完了するまで待ちます。
4. **rhdm-7.12.0-kogito-and-optaplanner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-employee-rostering/optaweb-employee-rostering-standalone/target** ディレクトリに移動します。
5. 以下のコマンドを実行して、従業員勤務 JAR ファイルを実行します。

```
java -jar quarkus-app/quarkus-run.jar
```



### 注記

**quarkus.datasource.db-kind** パラメーターの値は、ビルド時にデフォルトで **H2** に設定されます。別のデータベースを使用するには、スタンドアロンモジュールを再構築し、コマンドラインでデータベースタイプを指定します。たとえば、PostgreSQL データベースを使用するには、以下のコマンドを入力します。

```
mvn clean install -DskipTests -Dquarkus.profile=postgres
```

6. アプリケーションにアクセスするには、Web ブラウザーで **http://localhost:8080/** と入力します。

### 14.2.3. Maven を使用した従業員勤務表スターターアプリケーションの構築と実行

コマンドラインを使用して、従業員勤務表スターターアプリケーションを構築し、実行することができます。

この手順を使用する場合は、データはメモリーに保存されるため、サーバーが停止するとデータが失われます。データベースサーバーを使用してアプリケーションを構築し、実行して永続的に保存する方法は、[「コマンドラインからの永続データストレージを使用した従業員勤務表スターターアプリケーションの構築と実行」](#) を参照してください。

#### 前提条件

- [「デプロイメントファイルの準備」](#) の説明に従ってデプロイメントファイルを準備しておく。
- Java 開発キットがインストールされている。
- Maven がインストールされている。
- ホストからインターネットにアクセスできる。ビルドプロセスは、インターネットを使用して、外部のリポジトリから Maven パッケージをダウンロードします。

#### 手順

1. **optaweb-employee-rostering-backend** ディレクトリーに移動します。
2. 以下のコマンドを入力します。

```
mvn quarkus:dev
```

3. **optaweb-employee-rostering-frontend** ディレクトリーに移動します。
4. 以下のコマンドを入力します。

```
npm start
```



### 注記

**npm** を使用してサーバーを起動すると、**npm** によりコードの変更が監視されます。

5. アプリケーションにアクセスするには、Web ブラウザーで **http://localhost:3000/** と入力します。

## 14.2.4. コマンドラインからの永続データストレージを使用した従業員勤務表スターターアプリケーションの構築と実行

コマンドラインで従業員勤務表スターターアプリケーションを構築し、実行する場合には、データベースサーバーを指定して、永続的にデータを保存することができます。

### 前提条件

- 「[デプロイメントファイルの準備](#)」の説明に従ってデプロイメントファイルを準備しておく。
- Java 開発キットがインストールされている。
- Maven がインストールされている。
- ホストからインターネットにアクセスできる。ビルドプロセスは、インターネットを使用して、外部のリポジトリから Maven パッケージをダウンロードします。
- MySQL または PostgreSQL データベースサーバーがデプロイされている。

### 手順

1. 端末で、**optaweb-employee-rostering-standalone/target** ディレクトリーに移動します。
2. 以下のコマンドを実行して、従業員勤務 JAR ファイルを実行します。

```
java \
-Dquarkus.datasource.username=<DATABASE_USER> \
-Dquarkus.datasource.password=<DATABASE_PASSWORD> \
-Dquarkus.datasource.jdbc.url=<DATABASE_URL> \
-jar quarkus-app/quarkus-run.jar
```

上記の例で、以下のプレースホルダーを置き換えてください。

- **<DATABASE\_URL>**: データベースに接続する URL
- **<DATABASE\_USER>**: データベースに接続するユーザー
- **<DATABASE\_PASSWORD>**: **<DATABASE\_USER>** のパスワード



### 注記

**quarkus.datasource.db-kind** パラメーターの値は、ビルド時にデフォルトで **H2** に設定されます。別のデータベースを使用するには、スタンドアロンモジュールを再構築し、コマンドラインでデータベースタイプを指定します。たとえば、PostgreSQL データベースを使用するには、以下のコマンドを入力します。

**mvn clean install -DskipTests -Dquarkus.profile=postgres**



### 14.2.5. IntelliJ IDEA を使用した従業員勤務表スターターアプリケーションの構築および実行

IntelliJ IDEA を使用して、従業員勤務表スターターアプリケーションを構築し、実行することができます。

#### 前提条件

- [Employee Rostering](#) GitHub ページから従業員勤務表のソースコードをダウンロードしている。
- IntelliJ IDEA、Maven、および Node.js がインストールされている。
- ホストからインターネットにアクセスできる。ビルドプロセスは、インターネットを使用して、外部のリポジトリから Maven パッケージをダウンロードします。

#### 手順

1. IntelliJ IDEA を起動します。
2. IntelliJ IDEA メインメニューから **File** → **Open** を選択します。
3. アプリケーションソースの root ディレクトリーを選択し、**OK** をクリックします。
4. メインメニューから **Run** → **Edit Configurations** を選択します。
5. 表示されるウインドウで **Templates** を展開し、**Maven** を選択します。Maven サイドバーが表示されます。
6. Maven サイドバーの **Working Directory** メニューから **optaweb-employee-rostering-backend** を選択します。
7. コマンドライン で **mvn quarkus:dev** と入力します。
8. バックエンドを起動するには、**OK** をクリックします。
9. 端末で、**optaweb-employee-rostering-frontend** ディレクトリーに移動します。
10. 以下のコマンドを入力して、フロントエンドを起動します。

```
npm start
```

11. アプリケーションにアクセスするには、Web ブラウザーで **http://localhost:3000/** と入力します。

### 14.3. 従業員勤務表スターターアプリケーションのソースコードに関する概要

従業員勤務表スターターアプリケーションは、以下の主要コンポーネントで設定されています。

- Red Hat ビルドの OptaPlanner を使用して勤務表のロジックを実装し、REST API を提供する **backend**
- React を使用してユーザーインターフェイスを実装し、REST API で **backend** モジュールと対話する **frontend** モジュール

上記のコンポーネントを個別にビルドして使用することができます。特に、異なるユーザーインターフェイスを実装して、REST API でサーバーを呼び出すことができます。

主なコンポーネント 2 つに加え、従業員勤務表テンプレートには、ランダムなソースデータのジェネレーター (デモやテスト目的で便利) やベンチマークアプリケーションが含まれます。

## モジュールおよび主要なクラス

従業員勤務表テンプレートの Java ソースコードには複数の Maven モジュールが含まれます。これらのモジュールごとに、個別の Maven プロジェクトファイル (**pom.xml**) が含まれていますが、これは共通のプロジェクトで構築するために設計されています。

モジュールには、Java クラスなど複数のファイルが含まれます。このドキュメントでは、全モジュールと、従業員勤務表計算の主な情報を含むその他のファイルとクラスをリストします。

- **optawebpemployee-rostering-benchmark** モジュール: 乱数データを生成し、ソリューションをベンチマーク化する追加のアプリケーションが含まれます。
- **optaweb-employee-rostering-distribution** モジュール: README ファイルが含まれます。
- **optaweb-employee-rostering-docs** モジュール: ドキュメントファイルが含まれます。
- **optaweb-employee-rostering-frontend** モジュール: React で開発したユーザーインターフェイスを使用するクライアントアプリケーションが含まれます。
- **optaweb-employee-rostering-backend** モジュール: OptaPlanner を使用して勤務表の計算を行うサーバーアプリケーションが含まれます。
  - **src/main/java/org.optaweb.employee-rostering.service.roster/rosterGenerator.java**: デモおよびテスト目的でランダムな入力データを生成します。必要な入力データを変更する場合には、ジェネレーターも合わせて変更してください。
  - **src/main/java/org.optaweb.employee-rostering.domain.employee/EmployeeAvailability.java**: 従業員の空き情報を定義します。時間枠ごとに、従業員の空き状況と、従業員の希望する時間枠を指定できます。
  - **src/main/java/org.optaweb.employee-rostering.domain.employee/Employee.java**: 従業員を定義します。従業員には名前とスキル一覧があり、契約に従って仕事に従事します。スキルは、スキルオブジェクトで表現します。
  - **src/main/java/org.optaweb.employee-rostering.domain.roster/Roster.java**: 計算済みの勤務表情報を定義します。
  - **src/main/java/org.optaweb.employee-rostering.domain.shift/Shift.java**: 従業員を割り当て可能なシフトを定義します。シフトは、時間枠とスポットで定義します。たとえば、レストランでは、Kitchen のスポットで、2 月 20 日 8AM ~ 4PM の時間枠のシフトなどがあります。複数のシフトを、特定のスポットと時間枠に定義できます。今回の例では、このスポットと時間枠には複数の従業員が必要です。
  - **src/main/java/org.optaweb.employee-rostering.domain.skill/Skill.java**: 従業員に割り当て可能なスキルを定義します。
  - **src/main/java/org.optaweb.employee-rostering.domain.spot/Spot.java**: 従業員を配置可能なスポットを定義します。たとえば、Kitchen をスポットとして指定できます。
  - **src/main/java/org.optaweb.employee-rostering.domain.contract/Contract.java**: さまざまな期間の従業員の労働時間を制限する契約を定義します。

- **src/main/java/org.optaweb.employee rostering.domain.tenant.Tenant.java**: テナントを定義します。テナントごとに、独立したデータセットを表します。あるテナントのデータが変更されても、他のテナントには影響がありません。
- **\*View.java**: ドメインオブジェクト関連のクラス。他の情報から計算する値のセットを定義します。クライアントアプリケーションは、REST API 経由でこれらの値を読み取りますが、書き込みはできません。
- **\*Service.java**: サービスパッケージに配置されるインターフェイスで REST API を定義します。サーバーとクライアントアプリケーションのいずれも、これらのインターフェイスの実装を個別に定義します。
- **optaweb-employee-rostering-standalone** モジュール: スタンドアロンアプリケーションのアーセンブリの設定が含まれます。

## 14.4. 従業員勤務表スターターアプリケーションの変更

従業員勤務表スターターアプリケーションをニーズに合わせて変更するには、最適化プロセスを統括するルールを変更する必要があります。また、データ構造に必要なデータを含み、ルールに必要な計算が提供されるようにする必要があります。必要なデータがユーザーインターフェイスに存在しない場合は、ユーザーインターフェイスも変更する必要があります。

以下の手順では、従業員勤務表スターターアプリケーションの変更に関する一般的なアプローチを説明しています。

### 前提条件

- アプリケーションを正常にビルドするビルド環境がある。
- Java コードの読み取りと変更ができる。

### 手順

1. 必要な変更をプランニングします。以下の質問に答えてください。
  - 回避する **必要** がある追加のシナリオは何ですか。これらのシナリオは **ハード制約** です。
  - 可能な場合は、Optimizer が **回避しなければ** ならない追加のシナリオは何ですか。これらのシナリオは **ソフト制約** です。
  - それぞれのシナリオがソリューションで実行されるかどうかを計算するのに必要なデータは何ですか。
  - 既存のバージョンで入力した情報から取得可能なデータはどれですか。
  - ハードコードが可能なデータはどれですか。
  - ユーザー入力が必要なデータと、現在のバージョンで入力されていないデータはどれですか。
2. 必須データを現在のデータから計算するか、ハードコード化できる場合は、計算か、ハードコードを既存のビューまたはユーティリティークラスに追加します。データをサーバー側で計算する必要がある場合は、REST API エンドポイントを追加して読み込みます。
3. ユーザーが必須データを入力する必要がある場合は、データのエントリーを表現するクラスにそのデータを追加 (例: **Employee** クラス) し、データの読み取り、および書き込み用に REST

API エンドポイントを追加して、データ入力用にユーザーインターフェイスを変更してください。

4. 全データが利用できる場合は、ルールを変更します。変更の大半は、新規ルールを追加する必要があります。これらのルールは、**optaweb-employee-rostering-backend** モジュールの **src/main/java/org/optaweb/employee rostering/service/solver/EmployeeRosteringConstraintProvider.java** ファイルに配置されます。
5. アプリケーションの変更後に、ビルドして実行します。

## 第15章 RED HAT OPENSIFT CONTAINER PLATFORM での RED HAT ビルドの OPTAPLANNER のデプロイメントおよび使用: 従業員勤務表スターターアプリケーション

ビジネスルールの作成者は、Red Hat Decision Manager ディストリビューションに含まれる **optaweb-employee-rostering** スタータープロジェクトを OpenShift に簡単にデプロイして、Red Hat ビルドの OptaPlanner 機能をテストして操作できます。

### 前提条件

- デプロイした OpenShift 環境にアクセスできる。詳細は、使用する OpenShift 製品のドキュメンテーションを参照してください。

### 15.1. 従業員勤務表スターターアプリケーションの概要

従業員勤務表スターターアプリケーションは、組織内のさまざまな場所に従業員を割り当てます。たとえば、アプリケーションを使用して、病院での看護師のシフト、さまざまな場所での警備勤務シフト、作業者の組み立てラインのシフトを割り当てます。

従業員勤務表を最適化するには、多くの変数を考慮する必要があります。たとえば、業務が異なれば、求められるスキルが異なります。また、従業員の中には、特定の時間帯に勤務できない場合や、特定の時間帯での勤務を希望する場合があります。さらに、従業員によっては、1回に就業できる時間に制限がある契約を交わしている可能性があります。

このスターターアプリケーションの Red Hat ビルドの OptaPlanner ルールは、ハード制約およびソフト制約を使用します。最適化時に、従業員が勤務できない (または病欠の) 場合や、ある1つのシフト内の2つのスポットで働くことができない場合など、プランニングエンジンはハード制約に違反することができません。プランニングエンジンは、ソフト制約 (特定のシフトで勤務しないという従業員の希望など) に順守しようとしますが、最適なソリューションには違反が必要だと判断した場合は、違反することができます。

### 15.2. OPENSIFT での従業員勤務表スターターアプリケーションのインストールおよび起動

**runOnOpenShift.sh** スクリプトを使用して、従業員名簿スターターアプリケーションを Red Hat OpenShift Container Platform にデプロイします。**runOnOpenShift.sh** シェルスクリプトは、Red Hat Decision Manager 7.12.0Kogito および OptaPlanner 8 Decision Services クイックスタートディストリビューションで利用できます。

**runOnOpenShift.sh** スクリプトは、アプリケーションのソースコードをローカルでビルドおよびパッケージ化し、デプロイのために OpenShift 環境にアップロードします。このメソッドには、Java Development Kit、Apache Maven、および bash シェルコマンドラインが必要です。

#### 15.2.1. 提供されているスクリプトを使用したアプリケーションのデプロイ

従業員勤務表スターターアプリケーションは、提供されているスクリプトを使用して Red Hat OpenShift Container Platform にデプロイできます。このスクリプトは、アプリケーションのソースコードをローカルでビルドしてパッケージ化し、OpenShift 環境にアップロードしてデプロイします。

### 前提条件

- **oc** コマンドラインツールを使用して、対象の OpenShift 環境にログインしている。このツールの詳細については、[OpenShift Container Platform CLI リファレンス](#) を参照してください。

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスタマーポータル (ログインが必要) の [ソフトウェアダウンロード](#) ページから入手できます。
- Apache Maven 3.6 以降がインストールされている。Maven は [Apache Maven Project](#) の Web サイトから入手できます。
- **bash** シェル環境がローカルのマシンに用意されている。

## 手順

1. Red Hat カスタマーポータルの [Software Downloads](#) ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
  - Product: Decision Manager
  - バージョン: 7.12.0
2. Red Hat Decision Manager 7.12 Maven リポジトリ Kogito および OptaPlanner 8 Maven リポジトリ (**rhdm-7.12.0-kogito-maven-repository.zip**) ファイルをダウンロードします。
3. **rhdm-7.12.0-kogito-maven-repository.zip** ファイルを抽出します。
4. **rhdm-7.12.0-kogito-maven-repository / maven-repository** サブディレクトリの内容を **~/ .m2 / repository** ディレクトリにコピーします。
5. Red Hat カスタマーポータルの [Software Downloads](#) ページから **rhdm-7.12.0-kogito-and-optaplanner-quickstarts.zip** ファイルをダウンロードします。
6. ダウンロードしたアーカイブを展開します。
7. **optaweb-employee-rostering** フォルダーに移動します。
8. 従業員勤務表アプリケーションをビルドするには、以下のコマンドを実行します。

```
mvn clean install -DskipTests -DskipITs
```

9. OpenShift アカウントまたは Red Hat Code Ready Container インスタンスにログインします。以下の例では、**<account-url>** を OpenShift アカウントまたは Red Hat Code Ready Container インスタンスに、**<login-token>** をそのアカウントのログイントークンに置き換えます。

```
oc login <account-url> --token <login-token>
```

10. 従業員勤務表をホストする新規プロジェクトを作成します。

```
oc new-project optaweb-employee-rostering
```

11. プロビジョニングスクリプトを実行し、アプリケーションをビルドしてデプロイします。

```
./runOnOpenShift.sh
```

コンパイルとパッケージ化が完了するには最大 10 分かかります。これらのプロセスは、継続的にコマンドライン出力に進行状況を示します。

操作が完了したら、以下のメッセージが表示されます。**<URL>** は デプロイメントの URL に置き換えます。

You can access the application at <URL> once the deployment is done.

12. 先ほどの手順で使用した OpenShift アカウントまたは Red Hat Code Ready Container インスタンスの URL を入力して、デプロイしたアプリケーションにアクセスします。初回起動の場合は、OpenShift プラットフォームでの追加のビルドが実行されるため、最大で1分程かかります。



#### 注記

リンクをクリックしてから1分経過してもアプリケーションが表示されない場合は、ブラウザーページを強制的に更新してください。

## 15.3. 従業員勤務表スターターアプリケーションの使用

Web インターフェイスで、従業員勤務表アプリケーションを使用することができます。このインターフェイスは、ReactJS で開発します。また、REST API にアクセスして、必要に応じてカスタムのユーザーインターフェイスを構築することもできます。

### 15.3.1. ドラフトおよび公開期間

特定の時点で、アプリケーションを使用して、**ドラフト** 期間と呼ばれる期間の勤務表を作成できます。デフォルトでは、ドラフト期間は 3 週間です。

ドラフト期間の1週目に勤務表が最終版とされた場合に、勤務表を **公開** できます。この時点で、現在のドラフト期間の1週目の勤務表は、**公開** 期間になります。公開期間では勤務表は固定され、自動的に変更できなくなります (ただし、緊急の手動変更はまだ可能です)。この勤務表は従業員に配布され、この勤務表にあわせて予定を組むことができます。ドラフト期間は、1週間後に変更されます。

たとえば、ドラフト期間が9月1日から9月21日に設定されているとします。この期間には自動で従業員勤務表を作成できます。勤務表を公開したら、9月7日までの期間が公開されます。新規のドラフト期間は9月8日から28日です。

勤務表の公開に関する説明は、「[シフト勤務表の公開](#)」を参照してください。

### 15.3.2. ローテーションパターン

従業員勤務表アプリケーションは、シフトと従業員の **ローテーションパターン** をサポートします。

ローテーションパターンは、2日以上以上の期間を対象とするモデル期間です。このパターンは、特定の日付けには紐付けされません。

ローテーションの全日に対して時間バケットを作成できます。時間バケットにはすべて、シフトの時間を設定できます。必要に応じて、テンプレートに、シフトに割り当てる従業員名をデフォルトで含めることができます。

勤務表の公開時に、アプリケーションによりドラフト期間に新しい週が追加されます。この時点で、シフトおよび該当する場合はデフォルトの従業員名が、新しいドラフト期間にローテーションパターンからコピーされます。

ローテーションパターンの最後に到達すると、自動的に最初から開始されます。

週末のシフトパターンが平日と異なる場合は、1週間のローテーションパターンか、複数週間 (例: 14日、21日または28日) のローテーションパターンを使用してください。デフォルトの長さは28日です。常に同じ平日には同じパターンを繰り返して、別の平日に特定のシフトを設定できます。



ローテーションパターンの編集に関する説明は、「[ローテーションパターンの表示および編集](#)」を参照してください。

### 15.3.3. 従業員名簿テナント

従業員名簿アプリケーションは、複数のテナントをサポートします。各テナントは、入力や名簿の出力など、独立したデータのセットです。1つのテナントのデータを変更しても他のテナントへの影響はありません。テナントを切り替えて、独立したデータセットを複数使用できます。たとえば、複数の勤務地用に従業員の勤務表を複数作成できます。

インストール後には、工場や病院など、典型的な企業タイプを表す、サンプルテナントが複数存在します。これらのテナントのいずれかを選択して、ニーズに合わせて変更できます。また、新規テナントを作成して、白紙の状態からデータを入力できます。

#### 15.3.3.1. 従業員名簿テナントの変更

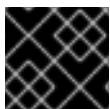
現在のテナントを変更できます。別のテナントを選択すると、表示されるすべての情報はこのテナントを参照し、行った変更はこのテナントにのみ影響します。

##### 手順

1. Employee Rostering アプリケーションの Web インターフェイスで、ブラウザーウィンドウの右上にあるテナントリストをクリックします。
2. リストからテナントを選択します。

#### 15.3.3.2. テナントの作成

新規テナントを作成して、白紙の状態からデータを入力できます。テナントの作成時には、複数のパラメーターを設定し、これらのパラメーターにより、アプリケーションがこのテナントの出力をどのように準備するかが決まります。



##### 重要

テナントパラメーターは、テナントの作成後に変更できません。

##### 手順

1. 従業員勤務表アプリケーション Web インターフェイスで新しいテナントを作成するには、ブラウザーウィンドウの右上隅で設定 (ギア) アイコンをクリックしてから **Add** をクリックします。
2. 以下の値を設定します。
  - **名前:** 新規テナントの名前。この名前は、テナントの一覧に表示されます。
  - **開始日のスケジュール:** 最初のドラフト期間の開始日。勤務表を公開すると、この日付が公開期間の開始日になります。対象の日付で平日の場合はそのまま、ドラフト期間の開始日、特定の公開期間、ローテーションパターンの初回使用日が継承されます。そのため、通常、開始日は週の初め (日曜か月曜) に設定すると最も便利です。
  - **ドラフト期間の長さ (日数):** ドラフト期間の長さ。ドラフト期間は、テナントの有効期限の長さと同じです。



- **公開通知期間 (日数):** 公開通知期間の長さ。従業員がシフトの時間をもとに個人の生活をプランニングできるように、最低でも指定の日数前に勤務表の最終版を公開するように促します。現在のバージョンではこの設定は有効ではありません。
- **公開期間の長さ (日数):** 勤務表を公開するたびに公開 (固定) される期間の長さ。現在のバージョンでは、この設定は 7 日に固定されています。
- **ローテーションの長さ (日数):** ローテーションパターンの長さ。
- **タイムゾーン:** 勤務表が適用される環境のタイムゾーン。このタイムゾーンは、ユーザーインターフェイスに表示される現在の日付を決定するのに使用します。

3. **Save** をクリックします。

テナントは空白データで作成します。

### 15.3.4. スキルの入力

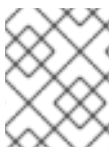
勤務表内の作業場所に必要となる **スキル** をすべて設定できます。たとえば、レストランの 24 時間シフトには、勤務人数やレストラン営業などの一般的なスキルに加え、調理、給仕、送迎、ホスティングサービスのスキルが必要になります。

#### 手順

1. 従業員名簿アプリケーションの Web インターフェイスで、**スキル** タブをクリックします。ブラウザーウィンドウの右上に、**1-15 of 34** など、現在表示可能なスキルの数が表示されます。＜ボタンおよび＞ボタンを使用して、リストの他のスキルを表示できます。

**Search** ボックスに、スキル名の一部を入力してスキルを検索できます。

2. 以下の手順を実行して、新規スキルを追加します。
  - a. **Add** をクリックします。
  - b. **Name** のテキストフィールドで新規スキル名を入力します。
  - c. 保存アイコンをクリックします。
3. スキル名を編集するには、スキルの横にある **Edit Skill** アイコン (鉛筆の形) をクリックします。
4. スキルを削除するには、スキルの横にある **Delete Skill** アイコン (ゴミ箱の形) をクリックします。



#### 注記

各テナントのスキルには一意の名前を指定する必要があります。スキルが従業員またはスポットに関連付けられている場合は、スキルを削除できません。

### 15.3.5. スポットの編集

ビジネス内のさまざまな業務を表す **スポット** のリストを入力する必要があります。レストランの場合では、スポットにはバー、食器片付けカウンター、受付カウンター、各種キッチンエリア、給仕エリア、オフィスなどが含まれます。

スポットごとに、**Skills** タブで入力したリストから必要なスキルを 1 つまたは複数選択できます。この

アプリケーションでは、スポットの全必要スキルを持つ従業員のみを、対象のスポットに割り当てます。スポットに必要なスキルがない場合、アプリケーションはこのスポットにどの従業員でも割り当てることができます。

## 手順

1. 従業員勤務表アプリケーションの Web インターフェイスでスポットを入力または変更するには、**Spots** タブをクリックします。**Search** ボックスに、スポット名の一部を入力してスポットを検索できます。
2. 以下の手順を実行して、新規スポットを追加します。
  - a. **Add Spot** をクリックします。
  - b. **Name** のテキストフィールドに新しいスポットの名前を入力します。
  - c. 必要に応じて、**Required skill set** のドロップダウンリストから1つまたは複数のスキルを選択します。
  - d. 保存アイコンをクリックします。
3. 名前と必要なスキルを編集するには、スポットの横にある **Edit Spot** アイコン (鉛筆の形) をクリックします。
4. スポットを削除するには、スポットの横にある **Delete Spot** アイコン (ゴミ箱の形) をクリックします。



## 注記

各テナント内では、一意のスポット名を使用する必要があります。スポットに対してシフトを作成している場合は、そのスポットを削除できません。

## 15.3.6. 契約リストの入力

ビジネスが従業員に使用する全種類の契約を入力する必要があります。

契約は、1日の最大就業時間、暦週、暦月、または暦年を決定します。

契約の作成時に、制約なしにすることも、制約を指定することも可能です。たとえば、パートタイムの従業員は週に 20 時間以上就業できない、フルタイムの従業員の就業時間は1日に 10 時間、1年に 1800 時間以内に制限するなどです。別の契約として、就業時間の制限なしなどを追加できます。

契約の就業時間の制限は、分単位で入力する必要があります。

## 手順

1. 従業員勤務表アプリケーションの Web インターフェイスで契約一覧を入力または変更するには、**Contracts** タブをクリックします。  
 ブラウザーウィンドウの右上に、**1-15 of 34** など、現在表示可能な契約数が表示されます。 < ボタンおよび > ボタンを使用して、リストの他の契約を表示できます。  
  
**Search** ボックスに、契約名の一部を入力して契約を検索できます。
2. 以下の手順を実行して、新しい契約を追加します。
  - a. **Add** をクリックします。

- b. **Name** のテキストフィールドに契約の名前を入力します。
  - c. **Maximum minutes** で、必要とされる時間の制限を入力します。
    - 従業員が1日に指定の時間以上就業してはいけない場合は、**Per Day** のチェックボックスを有効にして、このチェックボックスの横にあるフィールドに時間数を分単位で入力します。
    - 従業員が1週間に指定の時間以上就業してはいけない場合は、**Per Week** のチェックボックスを有効にして、このチェックボックスの横にあるフィールドに時間数を分単位で入力します。
    - 従業員が1ヶ月に指定の時間以上就業してはいけない場合は、**Per Month** のチェックボックスを有効にして、このチェックボックスの横にあるフィールドに時間数を分単位で入力します。
    - 従業員が1年に指定の時間以上就業してはいけない場合は、**Per Year** のチェックボックスを有効にして、このチェックボックスの横にあるフィールドに時間数を分単位で入力します。
  - d. 保存アイコンをクリックします。
3. 契約の名前と制限時間を編集するには、契約の名前の横にある **Edit Contract** アイコン (鉛筆の形) をクリックします。
  4. 契約を削除するには、契約の横にある **Delete Contract** アイコン (ゴミ箱の形) をクリックします。



#### 注記

各テナント内で、一意のコントラクト名を使用する必要があります。契約が従業員に割り当てられている場合は、削除できません。

### 15.3.7. 従業員リストの入力

このビジネスの全従業員、所有するスキル、適用する契約の一覧を入力する必要があります。このアプリケーションは、スキルや契約で指定された就業時間制限に合わせてスポットをこれらの従業員に割り当てます。

#### 手順

1. 従業員勤務表アプリケーションの Web インターフェイスで従業員一覧を入力または変更するには、**Employees** タブをクリックします。  
ブラウザーウィンドウの右上に、**1-15 of 34** など、現在表示可能な従業員の数が表示されます。＜ボタンおよび＞ボタンを使用して、リストの他の従業員を表示できます。

**Search** ボックスに、従業員名の一部を入力して従業員を検索できます。

2. 以下の手順を実行して、新規従業員を追加します。
  - a. **Add** をクリックします。
  - b. **Name** のテキストフィールドに新しい従業員の名前を入力します。
  - c. 必要に応じて、**Skill set** のドロップダウンリストから1つまたは複数のスキルを選択します。

- d. **Contract** のドロップダウンリストから契約を選択します。
- e. 保存アイコンをクリックします。
3. 従業員の名前とスキルを編集するには、従業員の名前の横にある **Edit Employee** アイコン (鉛筆の形) をクリックします。
4. 従業員を削除するには、従業員の名前の横にある **Delete Employee** アイコン (ゴミ箱の形) をクリックします。



#### 注記




各テナント内では、一意の従業員名を使用する必要があります。従業員に勤務日が割り当てられている場合は、その従業員を削除できません。

### 15.3.8. 従業員のアベイラビリティーの設定

特定の時間枠における従業員の空き時間を設定します。

従業員が特定の時間枠に **勤務できない (Unavailable)** 場合は、その時間枠に当てはまるシフトには割り当てられないようにすることができます (たとえば、病欠や休暇など)。**希望しない (Undesired)** および **希望 (Desired)** は、特定の時間枠における従業員の希望です。アプリケーションを使用して、可能な場合にこの希望に対応します。

#### 手順

1. 従業員勤務表アプリケーションの Web インターフェイスで従業員の空き時間一覧を表示または編集するには、**Availability Roster** タブをクリックします。  
ウィンドウの左上部分で、勤務表を表示する日付を確認できます。他の週を表示するには、**Week of** フィールドの横にある < ボタンと > ボタンを使用してください。または、日付フィールドをクリックして日付を変更し、この日付が含まれる週を表示します。
2. 従業員の勤務可能日エントリーを作成するには、スケジュールの空白のスペースをクリックしてから、従業員を選択します。最初は、全日分の **勤務不可** エントリーが作成されます。
3. 可用性エントリーを変更するには、エントリーをクリックします。以下の設定を変更することができます。
  - 日時 (**From** および **To**): 勤務可能エントリーに該当する時間枠
  - ステータス: ドロップダウンリストから **Unavailable**、**Desired**、または **Undesired** のステータスを選択できます。  
エントリーを保存するには **Apply** をクリックします。
4. 勤務可能のエントリーを削除するには、エントリーをクリックしてから **Delete availability** をクリックします。  
エントリーの上にマウスを移動してから、エントリーに表示されるアイコンの1つをクリックして、勤務可能のエントリーを変更または削除することもできます。
  -  アイコンをクリックして、エントリーのステータスを **Unavailable** に設定します。
  -  アイコンをクリックして、エントリーのステータスを **Undesired** に設定します。
  -  アイコンをクリックして、エントリーのステータスを **Desired** に設定します。

-  アイコンをクリックしてエントリーを削除します。



### 重要

従業員にすでにシフトが割り当てられており、このシフトの時間に勤務可能エントリーを作成または変更した場合には、この割当は自動的に変更されません。従業員のシフト勤務表を再度作り直さないと、新規または変更した勤務可能エントリーは適用されません。

## 15.3.9. シフト勤務表の表示および編集

シフト勤務表は、全スポットおよび、考えられる時間枠を示すテーブルです。

時間枠の1スポットに1名の従業員を割り当てる必要がある場合は、このスポットに対して **シフト** を1つ存在させる必要があります。スポットに同時に複数の従業員を割り当てる必要がある場合は、同じスポットと時間枠に複数のシフトを作成できます。

各シフトは、スポット (行) と期間 (列) が交差する長方形で示されます。

新しい時間がドラフト期間に追加された場合は、アプリケーションにより、ローテーションパターンからシフト (および、デフォルトの従業員がある場合には従業員) がドラフト期間の新しい部分にコピーされます。または、ドラフト期間のシフトを手動で追加して編集できます。

### 手順

1. 従業員勤務表アプリケーションの Web インターフェイスで勤務表を表示または編集するには、**Shift** タブをクリックします。  
ウィンドウの左上部分で、勤務表を表示する日付を確認できます。他の週を表示するには、**Week of** フィールドの横にある < ボタンと > ボタンを使用してください。または、日付フィールドをクリックして日付を変更し、この日付が含まれる週を表示します。
2. シフトを追加するには、スケジュールの空いているエリアをクリックします。アプリケーションは、クリックした場所から自動的にスロットと時間枠を判断し、シフトを追加します。
3. シフトを編集するには、シフトをクリックします。シフトには以下の値を設定できます。
  - **日時 (From および To):** シフトの正確な時間と期間
  - **Employee:** シフトに割り当てる従業員
  - **Pinned:** 従業員がシフトに **固定されている** かどうか。従業員が固定されている場合は、自動の従業員勤務表作成で、このシフトに対するこの従業員の割り当てを変更できません。固定されている従業員は、他のシフトに対して自動的に複製されません。変更を保存するには **Apply** をクリックします。
4. シフトを削除するには、シフトをクリックしてから **Delete shift** をクリックします。

## 15.3.10. 従業員のシフト勤務表の作成および表示

アプリケーションを使用して、全従業員に最適な勤務表を作成して表示できます。

### 手順

1. 従業員勤務表アプリケーションの Web インターフェイスで勤務表を表示または編集するには、**Shift** タブをクリックします。

2. 最適なシフト勤務表を作成するには、**Schedule** をクリックします。アプリケーションは、30 秒ほどで最適解を見つけ出します。

## 結果

操作が完了したら、Shift Roster ビューに最適なシフト勤務表が含まれています。ドラフト期間に新しい勤務表が作成されます。この操作では公開期間は変更されません。

ウィンドウの左上部分で、勤務表を表示する日付を確認できます。他の週を表示するには、**Week of** フィールドの横にある < ボタンと > ボタンを使用してください。または、日付フィールドをクリックして日付を変更し、この日付が含まれる週を表示します。

ドラフト期間では、シフトを表すボックスの境界線は、点線で表示されます。公開期間では、この境界線は実線になります。

シフトを表すボックスの色は、全シフトの制約ステータスを表します。

- 深緑: 一致するソフト制約。例: シフトが従業員の Desired (希望) 時間枠に当てはまる。
- 淡緑: 制約に違反がない。
- グレー: ソフト制約の違反。例: シフトが従業員の undesired (希望しない) 時間枠に当てはまる。
- 黄: 中間の制約違反。例: シフトに従業員が割り当てられていない。
- 赤: ハード制約の違反。例: 従業員に対して、同じ時間に 2 つのシフトが割り当てられている。

### 15.3.11. 従業員のシフトの表示

従業員ベースのテーブルで特定の従業員に割り当てられたシフトを表示できます。この情報は、シフト勤務表と同じですが、割り当てられたシフトを従業員に通知するのにより便利な表示形式になっています。

## 手順

従業員勤務表アプリケーションの Web インターフェイスで従業員およびシフトの表を表示するには、**Availability Roster** タブをクリックします。

ウィンドウの左上部分で、勤務表を表示する日付を確認できます。他の週を表示するには、**Week of** フィールドの横にある < ボタンと > ボタンを使用してください。または、日付フィールドをクリックして日付を変更し、この日付が含まれる週を表示します。

ブラウザーウィンドウの右上に、**1-10 of 34** など、現在表示可能な従業員の数が表示されます。数字の横にある < ボタンおよび > ボタンを使用して、リストの他の従業員を表示できます。

ドラフト期間では、シフトを表すボックスの境界線は、点線で表示されます。公開期間では、この境界線は実線になります。

### 15.3.12. シフト勤務表の公開

シフト勤務表を公開したら、ドラフト期間の最初の週が公開されます。従業員の勤務表の自動作成では、公開期間のシフト割当は変更されませんが、緊急の場合は手動で変更できます。これにより、ドラフト期間が 1 週間後ろにずれます。ドラフト期間と公開期間に関する詳細は、「[ドラフトおよび公開期間](#)」を参照してください。

## 手順

1. 従業員勤務表アプリケーションの Web インターフェイスで勤務表を表示または編集するには、**Shift** タブをクリックします。
2. ドラフト期間の最初の週に割り当てられたシフト勤務表をレビューして、許容範囲であるかを確認します。
3. **Publish** をクリックします。

### 15.3.13. ローテーションパターンの表示および編集

ローテーションパターンを使用すると、シフトを追加、移動、および削除できるため、従業員のリソースを効率的に管理できます。これは、時間バケットと座席によって定義されます。

The screenshot shows the OptaPlanner web interface with the 'Rotation' tab selected. The interface displays a grid of time buckets (6:00 AM-2:00 PM, 9:00 AM-5:00 PM, 2:00 PM-10:00 PM, 10:00 PM-6:00 AM) with employee stubs (A, B, C, E, F, G, L, P) assigned to specific days. A dropdown menu shows 'Anaesthetics' as the selected skill. A button 'Add New Time Bucket' is visible at the bottom left.

- 時間バケットは、特定のスポットまたは場所 (A) (たとえば 麻酔学) の 2 日以上にわたる時間枠 (たとえば午前 9 時から午後 5 時)、および必要なスキル (たとえば、射撃訓練) を説明します。
- 座席 (B) は、特定の時間バケット内の特定の日付に対する従業員割り当てです。
- 従業員のスタブは、時間バケットに割り当てることができる社員を表すアイコンです。従業員のスタブは、**Employee Stub List** の一覧に表示されています。

ローテーションパターンに関する詳細は、「[ローテーションパターン](#)」を参照してください。

#### 手順

1. 回転タブをクリックして、回転パターンを表示および編集します。
2. **Rotation** メニューからスポットを選択します。
3. **新しいタイムバケットの追加**をクリックします。[作業時間バケットの作成]ダイアログが表示されます。

4. 開始時間と終了時間を指定し、必要な追加のスキルを選択し、この時間バケットの日を選択して **Save** をクリックします。その時間バケットに割り当てられていない座席は、時間範囲別に編成された **Rotation** ページに表示されます。
5. 従業員のスタブ一覧をローテーションに追加するには、**Edit stub List** をクリックします。
6. **Edit Employee Stub List** ダイアログで **Add Employee** をクリックし、一覧から従業員を選択します。
7. このスタブリストに必要な社員をすべて追加し、**Save** をクリックします。この社員は、**Rotation** ページの時間バケットの上に表示されます。
8. 従業員アイコンをクリックして、従業員スタブリストから従業員を選択します。
9. 時間バケットの座席の上にマウスをクリックしてドラッグし、選択した従業員をそれらの座席に割り当てます。座席には従業員アイコンが表示されます。



### 注記

時間バケットは、1日に1の従業員しか割り当てることができません。複数の社員を同じ時間バケットに追加するには、時間バケットをコピーし、必要に応じて従業員の名前を変更します。

10. スケジュールをプロビジョニングするには、**Scheduling** をクリックし、ローテーションを作成したスポットを選択します。
11. **Provision** をクリックし、日付範囲を指定します。
12. このスケジュールに含まないスポットの選択を解除します。
13. 選択したスポットの横にある矢印をクリックし、スケジュールで使用しない時間バケットの選択を解除します。
14. **Provision Shifts** をクリックします。カレンダーには、時間バケットから生成されたシフトが反映されます。
15. 移動を変更するには、カレンダーで生成されたシフトをクリックします。



## 第16章 RED HAT ビルドの OPTAPLANNER の配送経路プランニングスターターアプリケーションのデプロイおよび使用

開発者は、OptaWeb 配送経路スターターアプリケーションを使用して、車両での配送を最適化できます。

### 前提条件

- OpenJDK (JDK) 11 がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスマーポータル (ログインが必要) の [ソフトウェアダウンロード](#) ページから入手できます。
- Apache Maven 3.6 以降がインストールされている。Maven は [Apache Maven Project](#) の Web サイトから入手できます。

### 16.1. OPTAWEB 配送経路

多くのビジネスは、各種貨物を輸送することを主な目的としています。これらのビジネスでは積荷の地点から目的地まで貨物を運送し、最も効率的な方法で車両を使用することを目指しています。主な目的の1つは、時間または距離のいずれかで測定される通過コストを最小限に抑えることです。

この種類の最適化問題は、運搬経路問題 (VRP: Vehicle Routing Problem) と呼ばれており、さまざまなバリエーションがあります。

Red Hat ビルドの OptaPlanner は、配送経路のバリエーションを多数解決して、ソリューションの例を提供します。OptaPlanner を使用すると、開発者は [制約プログラミング](#) 理論を学習するのではなく、ビジネスルールや要件のモデル化に焦点を当てることができます。OptaWeb 配送経路では、以下のような質問に解答するスターターアプリケーションを提供することで OptaPlanner の配送経路機能を拡張します。

- 距離と移動時間はどこから取得しますか？
- ソリューションをマップ上で視覚化するにはどうすればよいですか？
- クラウドで実行するアプリケーションを構築するにはどうすればよいですか？

OptaWeb 配送経路は OpenStreetMap (OSM) データファイルを使用します。OpenStreetMap の詳細は、[OpenStreetMap](#) の Web サイトを参照してください。

OptaWeb 配送経路を使用する場合は、次の定義を使用してください。

**地域:** OSM ファイルで表現される、地球上の地図の任意エリア。地域は、国、都市、大陸、頻繁にまとめて使用される複数の国などです。たとえば、DACH の地域にはドイツ (DE)、オーストリア (AT)、およびスイス (CH) が含まれます。

**国コード:** ISO-3166 標準により割り当てられた 2 文字のコード。国コードを使用して地理検索の結果を絞り込みます。複数の国にまたがる地域 (例: DACH 地域) を使用する場合があるため、OptaWeb 配送経路はこのような地域で地理検索の絞り込みができるように、国コードの一覧も使用できます。国コードの一覧については、[ISO 3166 Country Codes](#) を参照してください。

**地理検索:** 検索キーワードとして地域の住所や場所名を指定して、GPS の場所番号を結果として受け取るクエリーの種類。検索キーワードの一意性により、返される場所の数は異なります。大抵の場合、場所の名前は一意ではないため、作業地域の (複数の) 国の場所だけを含めることで、関連のない結果を除外します。

## 16.2. OPTAWEB 配送経路デプロイメントファイルのダウンロードおよびビルド

デプロイメントファイルをダウンロードし、準備してから、OptaWeb 配送経路の構築、デプロイを行う必要があります。

### 手順

1. Red Hat カスタマーポータルの [Software Downloads](#) ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
  - Product: Decision Manager
  - バージョン:7.12.0
2. Red Hat Decision Manager 7.12.0 Kogito および OptaPlanner 8 Decision Services クイックスタート (`rhdm-7.12.0-kogito-and-optaplanter-quickstarts.zip`) をダウンロードします。
3. `rhdm-7.12.0-kogito-and-optaplanter-quickstarts.zip` ファイルを展開します。
4. Red Hat Decision Manager 7.12 Maven リポジトリ Kogito および OptaPlanner 8 Maven リポジトリ (`rhdm-7.12.0-kogito-maven-repository.zip`) をダウンロードします。
5. `rhdm-7.12.0-kogito-maven-repository.zip` ファイルを抽出します。
6. `rhdm-7.12.0-kogito-maven-repository / maven-repository` サブディレクトリーの内容を `~/ .m2 / repository` ディレクトリーにコピーします。
7. `optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing` ディレクトリーに移動します。
8. 以下のコマンドを入力して OptaWeb 配送経路をビルドします。

```
mvn clean package -DskipTests
```

## 16.3. RUNLOCALLY.SH スクリプトを使用してローカルで OPTAWEB 配送経路を実行します。

Linux を使用する場合は、`runLocally.sh` の Bash スクリプトを使用して OptaWeb 配送経路を実行できます。



### 注記

`runLocally.sh` スクリプトは、MacOS では実行されません。`runLocally.sh` スクリプトを使用できない場合は、「[OptaWeb 配送経路の手動での設定および実行](#)」を参照してください。

`runLocally.sh` スクリプトは、以下の設定手順を自動化しますが、このスクリプトを使用しない場合は、この設定を手動で実行する必要があります。

- データディレクトリーを作成します。
- Geofabrik から選択した OpenStreetMap (OSM) ファイルをダウンロードします。
- ダウンロードした各 OSM ファイルに国コードを自動的に関連付けてみます。

- スタンドアロン JAR ファイルが存在しない場合は、プロジェクトをビルドします。
- OptaWeb 配送経路を起動するには、地域の引数を1つ指定するか、対話的に地域を選択します。

**runLocally.sh** スクリプトの実行に関する説明は、以下のセクションを参照してください。

- 「[クイックスタートモードで OptaWeb 配送経路の runLocally.sh スクリプトを実行します。](#)」
- 「[OptaWeb 配送経路の runLocally.sh スクリプトを対話モードで実行](#)」
- 「[OptaWeb 配送経路の runLocally.sh スクリプトを非対話モードで実行](#)」

### 16.3.1. クイックスタートモードで OptaWeb 配送経路の runLocally.sh スクリプトを実行します。

最も簡単な方法で OptaWeb 配送経路を使い始めるには、引数を指定せずに **runLocally.sh** スクリプトを実行します。

#### 前提条件

- OptaWeb 配送経路が「[OptaWeb 配送経路デプロイメントファイルのダウンロードおよびビルド](#)」の説明通りに、正常に Maven でビルドされている。
- インターネットを利用できる。

#### 手順

1. **rhdm-7.12.0-kogito-and-optaplaner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing** ディレクトリーで以下のコマンドを入力します。

```
./runLocally.sh
```

2. **.optaweb-vehicle-routing** ディレクトリーを作成するようにプロンプトが表示されたら、**y** と入力します。スクリプトの初回実行時に、このディレクトリーを作成するようにプロンプトが表示されます。
3. OSM ファイルのダウンロードのプロンプトが表示された場合は、**y** と入力します。このスクリプトの初回実行時に、OptaWeb 配送経路が Belgium OSM ファイルをダウンロードします。アプリケーションは、OSM ファイルのダウンロード後に起動します。
4. OptaWeb 配送経路のユーザーインターフェイスを表示するには、Web ブラウザーに以下の URL を入力します。

```
http://localhost:8080
```



#### 注記

このスクリプトを初回実行する場合は、GraphHopper で OSM ファイルをインポートして道路網のグラフとして保存する必要があるため、起動に数分かかります。次回から **runlocally.sh** スクリプトを実行する場合、読み込み時間ははるかに短縮されます。

#### 次のステップ

[「OptaWeb 配送経路の使用」](#)

### 16.3.2. OptaWeb 配送経路の runLocally.sh スクリプトを対話モードで実行

対話モードを使用して、ダウンロードした OSM ファイルと、各地域に割り当てられた国コードの一覧を表示します。対話モードを使用すると、Web サイトに移動してダウンロード先を選択せずに Geofabrik から追加の OSM ファイルをダウンロードできます。

#### 前提条件

- OptaWeb 配送経路が [「OptaWeb 配送経路デプロイメントファイルのダウンロードおよびビルド」](#) の説明通りに、正常に Maven でビルドされている。
- インターネットを利用できる。

#### 手順

1. **rhdm-7.12.0-kogito-and-optaplaner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing** ディレクトリーに移動します。
2. 以下のコマンドを入力して対話モードでスクリプトを実行します。

```
./runLocally.sh -i
```

3. **Your choice** のプロンプトで、**d** を入力してダウンロードメニューを表示します。以前にダウンロードした地域の一覧と、その後にダウンロード可能な地域の一覧が表示されます。
4. 任意: 以前にダウンロードした地域の一覧から地域を選択します。
  - a. ダウンロードした地域の一覧で地域に関連付けられた番号を入力します。
  - b. Enter キーを押します。
5. 任意: 地域をダウンロードします。
  - a. ダウンロードする地域に関連付けられている番号を入力します。たとえば、ヨーロッパの地図を選択するには、**5** と入力します。
  - b. 地図をダウンロードするには、**d** と入力して、Enter キーを押します。
  - c. 地図内の特定の地域をダウンロードするには、**e** と入力して、ダウンロードする地域に関連付けられている番号を入力して、Enter キーを押します。



#### サイズの大きい OSM ファイルの使用

欧州の個々の国や、アメリカの州など、小さい地域を使用すると、最適なユーザーエクスペリエンスが得られます。1GB 以上の OSM ファイルを使用するには、かなりのメモリーサイズを必要とし、初期処理に時間がかかります (最大では数時間)。

アプリケーションは、OSM ファイルのダウンロード後に起動します。

6. OptaWeb 配送経路のユーザーインターフェイスを表示するには、Web ブラウザーに以下の URL を入力します。

```
http://localhost:8080
```

## 次のステップ

[「OptaWeb 配送経路の使用」](#)

### 16.3.3. OptaWeb 配送経路の runLocally.sh スクリプトを非対話モードで実行

非対話モードで OptaWeb 配送経路を使用して、コマンド1つで OptaWeb 配送経路を起動し、先ほどダウンロードした OSM ファイルを追加します。これは、地域間を素早く切り替える場合や、デモを行う場合に便利です。

#### 前提条件

- OptaWeb 配送経路が [「OptaWeb 配送経路デプロイメントファイルのダウンロードおよびビルド」](#) の説明通りに、正常に Maven でビルドされている。
- 使用する地域の OSM ファイルがダウンロードされている。OSM ファイルのダウンロードに関する詳細は、[「OptaWeb 配送経路の runLocally.sh スクリプトを対話モードで実行」](#) を参照してください。
- インターネットを利用できる。

#### 手順

1. **rhdm-7.12.0-kogito-and-optaplanner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing** ディレクトリーに移動します。
2. 以下のコマンドを実行します。ここでは、**<OSM\_FILE\_NAME>** は先ほどダウンロードした OSM ファイルに置き換えます。

```
./runLocally.sh <OSM_FILE_NAME>
```

## 次のステップ

[「OptaWeb 配送経路の使用」](#)

### 16.3.4. データディレクトリーの更新

別のデータディレクトリーを使用する場合は、OptaWeb 配送経路が使用するディレクトリーとは異なるデータディレクトリーに更新できます。デフォルトのデータディレクトリーは **\$HOME/.optaweb-vehicle-routing** です。

#### 前提条件

- OptaWeb 配送経路が [「OptaWeb 配送経路デプロイメントファイルのダウンロードおよびビルド」](#) の説明通りに、正常に Maven でビルドされている。

#### 手順

- 別のデータディレクトリーを使用するには、現在のデータディレクトリーに、**.DATA\_DIR\_LAST** ファイルへのディレクトリーの絶対パスを指定します。

- 地域に関連付けられている国コードを変更するには、現在のデータディレクトリーの **country\_codes** ディレクトリーにある対応のファイルを編集します。  
たとえば、スコットランドの OSM ファイルをダウンロードし、スクリプトで国コードを推測できなかった場合に、**country\_codes/scotland-latest** のコンテンツを GB に設定します。
- リージョンを削除するには、データディレクトリーの **openstreetmap** ディレクトリーにある対応の OSM ファイルを削除し、**graphhopper** ディレクトリーで地域のディレクトリーを削除します。

## 16.4. OPTAWEB 配送経路の手動での設定および実行

OptaWeb 配送経路を最も簡単な方法で実行するには、**runlocally.sh** スクリプトを使用します。ただし、お使いのシステムでバッシュを利用できない場合は、**runlocally.sh** スクリプトが実行する手順を手動で完了してください。

### 前提条件

- OptaWeb 配送経路が「[OptaWeb 配送経路デプロイメントファイルのダウンロードおよびビルド](#)」の説明通りに、正常に Maven でビルドされている。
- インターネットを利用できる。

### 手順

1. 経路データをダウンロードします。

経路エンジンでは、車両が場所間の移動にかかる時間を計算するのに地理データが必要です。ローカルのファイルシステムに OpenStreetMap (OSM) データファイルをダウンロードして保存してから、OptaWeb 配送経路を実行する必要があります。



#### 注記

OSM データファイルのサイズは通常 100 MB から 1 GB の間となり、ダウンロードに時間がかかるため、OptaWeb 配送経路アプリケーションをビルドまたは起動する前にこれらのファイルをダウンロードすることをお勧めします。

- a. Web ブラウザーで <http://download.geofabrik.de/> を開きます。
  - b. **Sub Region** リストで地域 (ヨーロッパなど) をクリックします。サブ地域ページが開きます。
  - c. **Sub Regions** の表で、国 (ベルギーなど) の OSM ファイル (**.osm.pbf**) をダウンロードします。
2. データのディレクトリー構造を作成します。  
OptaWeb 配送経路では、ファイルシステム上の複数の種類のデータを読み取り、書き込みます。**openstreetmap** ディレクトリーから OSM (OpenStreetMap) ファイルを読み取り、道路網グラフを **graphhopper** ディレクトリーに書き込み、**db** ディレクトリーでユーザーデータを永続化します。このデータをすべて格納する専用のディレクトリーを新たに作成して、今後簡単に新しいバージョンの OptaWeb 配送経路にアップグレードして、以前に作成したデータをそのまま使用できるようにします。
    - a. **\$HOME/.optaweb-vehicle-routing** ディレクトリーを作成します。
    - b. **\$HOME/.optaweb-vehicle-routing** ディレクトリーに **openstreetmap** ディレクトリーを作成します。



```
$HOME/.optaweb-vehicle-routing
└─ openstreetmap
```

- c. ダウンロードした OSM ファイル (**.osm.pbf** の拡張子が付いたファイル) をすべて **openstreetmap** ディレクトリーに移動します。  
残りのディレクトリー構造は、OptaWeb 配送経路アプリケーションにより、初回実行時に作成されます。作成後のディレクトリー構造は以下のとおりです。

```
$HOME/.optaweb-vehicle-routing
├─ db
│   └─ vrp.mv.db
├─ graphhopper
│   └─ belgium-latest
└─ openstreetmap
    └─ belgium-latest.osm.pbf
```

3. **rhdm-7.12.0-kogito-and-optaplanner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing/optaweb-vehicle-routing-standalone/target** ディレクトリーに移動します。
4. OptaWeb 配送経路を実行するには、以下のコマンドを実行します。

```
java \
-Dapp.demo.data-set-dir=$HOME/.optaweb-vehicle-routing/dataset \
-Dapp.persistence.h2-dir=$HOME/.optaweb-vehicle-routing/db \
-Dapp.routing.gh-dir=$HOME/.optaweb-vehicle-routing/graphhopper \
-Dapp.routing.osm-dir=$HOME/.optaweb-vehicle-routing/openstreetmap \
-Dapp.routing.osm-file=<OSM_FILE_NAME> \
-Dapp.region.country-codes=<COUNTRY_CODE_LIST> \
-jar quarkus-app/quarkus-run.jar
```

このコマンドでは、以下の変数を置き換えてください。

- **<OSM\_FILE\_NAME>**: 以前にダウンロードした地域で、使用予定の地域の OSM ファイル。
- **<COUNTRY\_CODE\_LIST>**: 地理検索クエリーの絞り込みに使用するコンマ区切りの国コード一覧。国コードの一覧については、[ISO 3166 Country Codes](#) を参照してください。アプリケーションは、OSM ファイルのダウンロード後に起動します。

以下の例では、OptaWeb 配送経路は中央アメリカの OSM の地図 (**central-america-latest.osm.pbf**) をダウンロードして、ベリーズ (BZ) とグアテマラ (GT) の国で検索を行います。

```
java \
-Dapp.demo.data-set-dir=$HOME/.optaweb-vehicle-routing/dataset \
-Dapp.persistence.h2-dir=$HOME/.optaweb-vehicle-routing/db \
-Dapp.routing.gh-dir=$HOME/.optaweb-vehicle-routing/graphhopper \
-Dapp.routing.osm-dir=$HOME/.optaweb-vehicle-routing/openstreetmap \
-Dapp.routing.osm-file=central-america-latest.osm.pbf \
-Dapp.region.country-codes=BZ,GT \
-jar quarkus-app/quarkus-run.jar
```

5. OptaWeb 配送経路のユーザーインターフェイスを表示するには、Web ブラウザーに以下の URL を入力します。

```
http://localhost:8080
```

## 次のステップ

[「OptaWeb 配送経路の使用」](#)

## 16.5. RED HAT OPENSIFT CONTAINER PLATFORM での OPTAWEB 配送経路の実行

Linux を使用する場合には、Bash スクリプト **runOnOpenShift.sh** を使用して、Red Hat OpenShift Container Platform に OptaWeb 配送経路をインストールできます。



### 注記

MacOS では **runOnOpenShift.sh** スクリプトは実行されません。

## 前提条件

- OpenShift クラスターにアクセスができ、OpenShift コマンドラインインターフェイス (**oc**) がインストールされている。Red Hat OpenShift Container Platform の情報は、[OpenShift Container Platform のインストール](#) を参照してください。
- OptaWeb 配送経路が [「OptaWeb 配送経路デプロイメントファイルのダウンロードおよびビルド」](#) の説明通りに、正常に Maven でビルドされている。
- インターネットを利用できる。

## 手順

1. Red Hat OpenShift Container Platform クラスターにログインするか、このクラスターを起動します。
  - a. 以下のコマンドを入力します。**<PROJECT\_NAME>** は新規プロジェクト名に置き換えます。

```
oc new-project <PROJECT_NAME>
```

- b. 必要に応じて、**rhdm-7.12.0-kogito-and-optaplanner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing** ディレクトリーに移動します。
- c. 以下のコマンドを入力して、**runOnOpenShift.sh** スクリプトを実行し、OpenStreetMap (OSM) ファイルをダウンロードします。

```
./runOnOpenShift.sh <OSM_FILE_NAME> <COUNTRY_CODE_LIST>  
<OSM_FILE_DOWNLOAD_URL>
```

このコマンドでは、以下の変数を置き換えてください。

- **<OSM\_FILE\_NAME>**: **<OSM\_FILE\_DOWNLOAD\_URL>** からダウンロードしたファイルの名前。



- **<COUNTRY\_CODE\_LIST>**: 地理検索クエリーの絞り込みに使用するコンマ区切りの国コード一覧。国コードの一覧については、[ISO 3166 Country Codes](#)を参照してください。
- **<OSM\_FILE\_DOWNLOAD\_URL>**: OpenShift からアクセス可能な、PBF 形式の OSM データの URL。このファイルは、バックエンドの起動中にダウンロードされ、**/deployments/local/<OSM\_FILE\_NAME>** として保存されます。  
以下の例では、OptaWeb 配送経路は中央アメリカの OSM の地図 (**central-america-latest.osm.pbf**) をダウンロードして、ベリーズ (BZ) とグアテマラ (GT) の国で検索を行います。

```
./runOnOpenShift.sh central-america-latest.osm.pbf BZ,GT
http://download.geofabrik.de/europe/central-america-latest.osm.pbf
```



#### 注記

**runOnOpenShift.sh** スクリプトのヘルプを参照するには、**./runOnOpenShift.sh --help** と入力します。

### 16.5.1. デプロイされた OptaWeb 配送経路アプリケーションをローカル変更で更新

Red Hat OpenShift Container Platform に OptaWeb 配送経路アプリケーションをデプロイした後に、バックエンドとフロントエンドを更新できます。

#### 前提条件

- OptaWeb 配送経路が Maven で正常にビルドされ、OpenShift にデプロイされている。

#### 手順

- バックエンドを更新するには、次の手順を実行します。
  1. ソースコードを変更し、Maven でバックエンドモジュールをビルドします。
  2. **rhdm-7.12.0-kogito-and-optaplanner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing** ディレクトリーに移動します。
  3. 以下のコマンドを入力して、OpenShift ビルドを起動します。

```
oc start-build backend --from-dir=. --follow
```

- フロントエンドを更新するには、次の手順を実行します。
  1. ソースコードを変更して、**npm** ユーティリティーでフロントエンドモジュールをビルドします。
  2. **sources/optaweb-vehicle-routing-frontend** ディレクトリーに移動します。
  3. 以下のコマンドを入力して、OpenShift ビルドを起動します。

```
oc start-build frontend --from-dir=docker --follow
```

#### 次のステップ

[「OptaWeb 配送経路の使用」](#)

## 16.6. OPTAWEB 配送経路の使用

OptaWeb 配送経路アプリケーションでは、地図に場所の数をマークできます。最初の場所はデポ (配送拠点) であることを前提とします。車両はこのデポから、マークの付いた他のすべての場所に商品を配送する必要があります。

車両の数、および全車両の積載容量を設定できますが、経路では全車両を使用する保証はありません。ただし、そのルートがすべての車両に使用されるとは限りません。アプリケーションは、最適なルートに必要な数だけ車両を使用します。

現在のバージョンには、一定の制限があります。

- 1つの場所に配送するたびに、車両の積載量が1ポイント消費されます。たとえば、積載量が10の車両は、デポに戻るまでに最大10箇所まで訪問できます。
- 車両や場所にカスタム名を設定することはできません。

### 16.6.1. 経路の作成

最適な経路を作成するには、OptaWeb 配送経路ユーザーインターフェイスの **Demo** タブを使用します。

#### 前提条件

- OptaWeb 配送経路が実行されており、ユーザーインターフェイスにアクセスできる。

#### 手順

1. OptaWeb 配送経路では、**Demo** をクリックして **Demo** タブを開きます。
2. 地図上の青いプラス/マイナスボタンを使用して車両数を設定します。デフォルトでは、車両ごとの積載量は10となっています。
3. 必要に応じて、地図上の四角の中にあるプラスボタンを使用して、拡大します。



#### 注記

地図の拡大にダブルクリックを使用しないでください。ダブルクリックすると、場所が作成されます。

4. デポの場所をクリックします。
5. 配送ポイントについては、地図の他の場所をクリックします。
6. 場所を削除する場合:
  - a. 削除する場所の上のマウスをかざして、場所の名前を表示します。
  - b. 画面の左側にある一覧でその場所名を検索します。
  - c. 名前の横にある X アイコンをクリックします。

場所を追加/削除したり、車両数を変更したりするたびに、アプリケーションは新しい最適経路を作成して表示します。そのソリューションで複数の車両を使用する場合、アプリケーションは、全車両の経路を別の色で表示します。

### 16.6.2. その他の情報の表示と設定

OptaWeb 配送経路の他のタブを使用し、追加の情報を表示して設定できます。

#### 前提条件

- OptaWeb 配送経路が実行されており、ユーザーインターフェイスにアクセスできる。

#### 手順

- **Vehicles** タブをクリックして、車両の表示、追加、削除や、全車両の積載量の設定が可能です。
- **Visits** タブをクリックし、場所を表示して削除します。
- **Route** タブをクリックして、各車両を選択して、選択した車両の経路を表示します。

### 16.6.3. OptaWeb 配送経路でのカスタムデータセットの作成

ベルギー内の複数の大都市を含むデモデータセットが組み込まれています。**Load demo** メニューで他のデモを利用する場合は、ご自身のデータセットを用意します。

#### 手順

1. OptaWeb 配送経路で、地図をクリックするか、地理検索を使用して、デポ1つと、訪問数1つ以上を追加します。
2. **Export** をクリックして、データセットディレクトリーにファイルを保存します。



#### 注記

データセットディレクトリーは、**app.demo.data-set-dir** プロパティーで指定したディレクトリーです。

アプリケーションが **runLocally.sh** スクリプト経由で実行されている場合は、データセットディレクトリーが **\$HOME/.optaweb-vehicle-routing/dataset** に設定されます。

それ以外の場合は、プロパティー **application.properties** ファイルから取得し、**rhdm-7.12.0-kogito-and-optaplanner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing/optaweb-vehicle-routing-standalone/target/local/dataset** にデフォルト設定されます。

**app.demo.data-set-dir** プロパティーを編集して、別のデータディレクトリーを指定できます。

3. YAML ファイルを編集して、データセットの一意名を選択します。
4. バックエンドを再起動します。

バックエンドを再起動した後に、**Load demo** メニューにデータセットディレクトリーのファイルが表示されます。

### 16.6.4. OptaWeb 配送経路のトラブルシューティング

OptaWeb 配送経路で予期せぬ動作をする場合は、以下の手順に従い、トラブルシューティングを行います。

### 前提条件

- OptaWeb 配送経路は実行されるが、予期せぬ動作をする。

### 手順

1. 問題を特定するには、バックエンドの端末出力ログを確認します。
2. 問題を解決するには、バックエンドデータベースを削除します。
  - a. バックエンドの端末ウィンドウで Ctrl+C を押して、バックエンドを停止します。
  - b. **optaweb-vehicle-routing/optaweb-vehicle-routing-backend/local/db** ディレクトリを削除します。
  - c. OptaWeb 配送経路を再起動します。

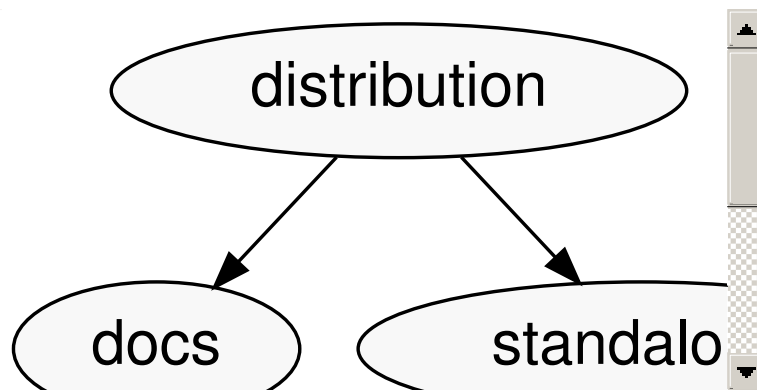
## 16.7. OPTAWEB 配送経路の開発ガイド

本セクションでは、開発モードでバックエンドモジュールおよびフロントエンドモジュールを設定して実行する方法を説明します。

### 16.7.1. OptaWeb 配送経路プロジェクトの構造

OptaWeb 配送経路プロジェクトは、マルチモジュールの Maven プロジェクトです。

図16.1 モジュール依存関係ツリーの図



バックエンドとフロントエンドモジュールは、モジュールツリーの一番下にあります。これらのモジュールには、アプリケーションのソースコードが含まれます。

スタンドアロンモジュールは、バックエンドとフロントエンドを1つの実行可能な JAR ファイルに統合するアセンブリーモジュールです。

ディストリビューションモジュールは、最終的なアセンブリーの手順を表します。このモジュールは、スタンドアロンアプリケーションと、ドキュメントを受けとり、配信しやすいようにアーカイブとしてラッピングします。

バックエンドとフロントエンドは、別にビルドとデプロイが可能な、別個のプロジェクトです。実際には、全く異なる言語で記述され、異なるツールでビルドされています。どちらのプロジェクトでも、コードの変更からアプリケーションの実行までの対応時間を短くし、最新の開発体験ができるようにす

るツールが含まれています。

次のセクションでは、バックエンドとフロントエンドのプロジェクトを開発モードで実行する方法を説明します。

### 16.7.2. OptaWeb 配送経路のバックエンドモジュール

バックエンドモジュールには、Red Hat ビルドの OptaPlanner を使用して配送経路を最適化するサーバー側のアプリケーションが含まれています。最適化は CPU を集中的に使用する計算であり、最大限に能力を発揮するには I/O 操作を回避する必要があります。移動コスト (時間または距離) を最小限に抑えることが主な目的の1つであるため、OptaWeb 配送経路では RAM メモリーに移動コストの情報を保持します。OptaPlanner は、解を出す時に、ユーザーが入力した全場所間の移動コストを把握しておく必要があります。この情報は、**距離行列** と呼ばれる構造に保存されます。

新しい場所を入力すると、OptaWeb 配送経路は新しい場所と、それ以外でこれまでに入力されたすべての場所との間の移動コストを計算して、距離行列にその移動コストを保存します。移動コストの計算は、[GraphHopper](#) の経路エンジンで実行されます。

バックエンドモジュールは、以下のような機能を追加で実装します。

- 永続性
- フロントエンドの WebSocket 接続
- データセットの読み込み、エクスポート、およびインポート

バックエンドコードのアーキテクチャーの詳細は、「[OptaWeb 配送経路のバックエンドアーキテクチャー](#)」を参照してください。

次のセクションでは、開発モードでバックエンドを設定して実行する方法を説明します。

#### 16.7.2.1. OptaWeb 配送経路のバックエンドモジュールの実行

バックエンドモジュールは、Quarkus 開発モードで実行できます。

##### 前提条件

- OptaWeb 配送経路が「[OptaWeb 配送経路の手動での設定および実行](#)」の説明どおりに設定されている。

##### 手順

1. `rhdm-7.12.0-kogito-and-optaplanner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing/optaweb-vehicle-routing-backend` ディレクトリーに移動します。
2. 開発モードでバックエンドを実行するには、次のコマンドを入力します。

```
mvn compile quarkus:dev
```

#### 16.7.2.2. IntelliJ IDEA Ultimate からの OptaWeb 配送経路バックエンドモジュールの実行

IntelliJ IDEA Ultimate を使用して OptaWeb 配送経路バックエンドモジュールを実行し、プロジェクトの開発を簡素化できます。IntelliJ IDEA Ultimate には、Quarkus プラグインが同梱されており、Quarkus フレームワークを使用するモジュールの実行設定を自動的に作成します。

## 手順

`optaweb-vehicle-routing-backend` の設定を使用して、バックエンドを実行します。

## 関連情報

詳細は、[Quarkus アプリケーションのビルド](#) を参照してください。

### 16.7.2.3. Quarkus 開発モード

開発モードでは、バックエンドのソースコードまたは設定が変更された場合に、フロントエンドが実行されるブラウザタブを更新すると、バックエンドが自動的に再起動します。

[Quarkus 開発モード](#) の詳細を確認してください。

### 16.7.2.4. OptaWeb 配送経路バックエンドモジュールのシステムプロパティーの値の変更

OptaWeb 配送経路バックエンドモジュールのデフォルトのシステムプロパティー値を一時的または永続的に上書きできます。

OptaWeb 配送経路バックエンドモジュールのプロパティーは `/src/main/resources/application.properties` ファイルに保存されます。このファイルはバージョン管理されます。このファイルを使用してデフォルトの設定プロパティーの値を永続的に保存し、Quarkus プロファイルを定義します。

## 前提条件

- OptaWeb 配送経路のスターターアプリケーションをダウンロードして展開している。詳細は、「[OptaWeb 配送経路デプロイメントファイルのダウンロードおよびビルド](#)」を参照してください。

## 手順

- デフォルトのシステムプロパティー値を一時的に上書きするには、`mvn` または `java` コマンドの実行時に、`-D<PROPERTY>=<VALUE>` 引数を追加します。`<PROPERTY>` は変更するプロパティーの名前、`<VALUE>` はそのプロパティーに一時的に割り当てる値に置き換えます。以下の例は、Maven を使用して `dev` モードで Quarkus プロジェクトをコンパイルした場合に、`quarkus.http.port` システムプロパティーの値を `8181` に一時的に変更する方法を示しています。

```
mvn compile quarkus:dev -Dquarkus.http.port=8181
```

これを実行すると、`/src/main/resources/application.properties` ファイルに保存されているプロパティーの値が一時的に変更されます。

- 開発環境に固有の設定を保存する場合など設定値を永続的に変更するには、`env-example` ファイルの内容を `optaweb-vehicle-routing-backend/.env` ファイルにコピーします。このファイルはバージョン管理には含まれないので、リポジトリのクローン時には存在しません。Git の作業ツリーに影響を与えずに、`.env` ファイルで変更を加えることができます。

## 関連情報

OptaWeb 配送経路の設定プロパティーに関する完全一覧は、「[OptaWeb 配送経路のバックエンド設定プロパティー](#)」を参照してください。

### 16.7.2.5. OptaWeb 配送経路のバックエンドログ

OptaWeb 配送経路は、SLF4J API と Logback をロギングフレームワークとして使用します。詳細は、[Quarkus のロギングの設定](#) を参照してください。

### 16.7.3. OptaWeb 配送経路のフロントエンドモジュールの操作

フロントエンドのプロジェクトは、[Create React App](#) でブートストラップされました。Create React App には、開発や、実稼働環境でアプリケーションをビルドしやすくするためのスクリプトや依存関係が多数含まれています。

#### 前提条件

- OptaWeb 配送経路のスターターアプリケーションをダウンロードして展開している。詳細は、「[OptaWeb 配送経路デプロイメントファイルのダウンロードおよびビルド](#)」を参照してください。

#### 手順

1. Fedora で、次のコマンドを入力して開発環境を設定します。

```
sudo dnf install npm
```

npm のインストールの詳細については、[Downloading and installing Node.js and npm](#) を参照してください。

2. **rhdm-7.12.0-kogito-and-optaplanner-quickstarts/optaweb-8.11.1.Final-redhat-00006/optaweb-vehicle-routing/optaweb-vehicle-routing-frontend** ディレクトリーに移動します。
3. **npm** の依存関係をインストールします。

```
npm install
```

Maven とは違い、**npm** パッケージマネージャーは、**npm install** を実行した場合にのみ、プロジェクトディレクトリーの **node\_modules** に依存関係をインストールします。**package.json** にリストされている依存関係が変更されると、master ブランチに変更をプルした時点で、**npm install** を実行してから開発サーバーを実行する必要があります。

4. 以下のコマンドを入力して、開発サーバーを実行します。

```
npm start
```

5. 自動的に表示されない場合には、Web ブラウザーで **http://localhost:3000/** を開きます。デフォルトでは **npm start** コマンドは、デフォルトのブラウザーでこの URL を開こうとします。



#### 注記

**npm start** コマンドで、実行するたびに新規ブラウザーのタブを開かないようにするには、**BROWSER=none** 環境変数をエクスポートします。**.env.local** ファイルを使用して、この設定を永続化します。これには、以下のコマンドを実行します。

```
echo BROWSER=none >> .env.local
```

ブラウザーは、フロントエンドのソースコードを変更するたびにページを更新します。端末で実行する開発サーバーのプロセスは、これらの変更を取得し、コンパイルエラーと lint エラーをコンソールに出力します。

- 以下のコマンドを入力して、テストを実行します。

```
npm test
```

- REACT\_APP\_BACKEND\_URL** 環境変数の値を変更して、**npm start**、**npm run build** などの実行時に、**npm** が使用するバックエンドプロジェクトの場所を指定します。

```
REACT_APP_BACKEND_URL=http://10.0.0.123:8081
```



### 注記

環境変数は、**npm** のビルドプロセス中は JavaScript バンドル内でハードコード化されるため、バックエンドの場所を指定してから、フロントエンドをビルドしてデプロイする必要があります。

React 環境変数の詳細は、[Adding Custom Environment Variables](#) を参照してください。

- フロントエンドをビルドするには、以下のコマンドのいずれか1つを実行します。

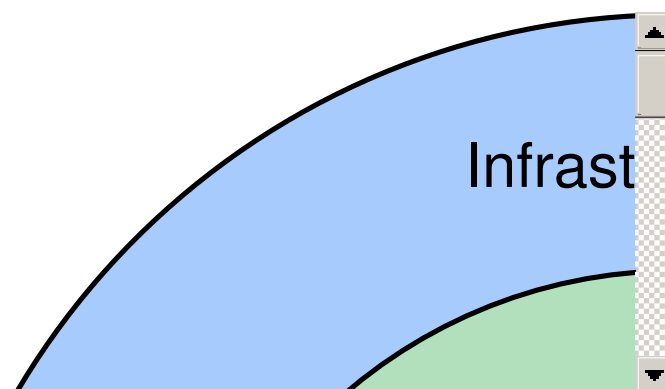
```
./mvnw install
```

```
mvn install
```

## 16.8. OPTAWEB 配送経路のバックエンドアーキテクチャー

ドメインモデルおよびユースケースは、アプリケーションには必要不可欠です。OptaWeb 配送経路ドメインモデルは、アーキテクチャーの中心にあり、その周りにユースケースを埋め込むアプリケーション層があります。経路最適化、距離計算、永続化、ネットワーク通信などの機能は実装の詳細とみなされ、アーキテクチャーの一番外側に配置されます。

図16.2 アプリケーション層の図

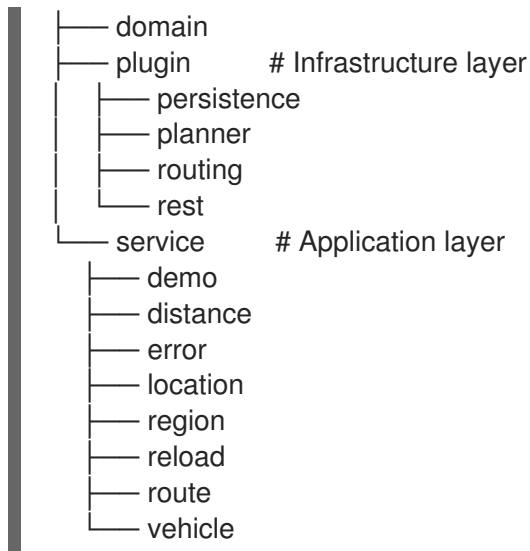


### 16.8.1. コードの編成

以前の図で示されるように、バックエンドコードは3つの層で整理されます。

```
org.optaweb.vehiclerouting
```





**service** パッケージには、ユースケースを実装するアプリケーション層が含まれます。**plugin** パッケージにはインフラストラクチャー層が含まれます。

各層のコードは、さらに機能別に編成されます。つまり、各サービスまたはプラグインに独自のパッケージがあります。

### 16.8.2. 依存関係ルール

コンパイル時間の依存関係は、外層から中心に向けてのみ可能です。このルールに従うことで、ドメインモデルを、基盤となるフレームワークや、他の実装詳細から独立させ、ビジネスエンティティの動作をより正確にモデル化できます。プレゼンテーションや永続性を周辺に押し出すことで、ビジネスエンティティとユースケースの動作をより簡単にテストできます。

ドメインには依存関係はありません。

サービスはドメインにだけ依存します。サービスが(データベースまたはクライアントに)結果を送信する必要がある場合には、出力境界インターフェイスを使用します。実装は [contexts and dependency injection](#) (CDI) コンテナで注入されます。

プラグインは、2つの方法でサービスに依存します。1つ目は、ユーザー入力や最適化エンジンによる経路の更新など、イベントを基にサービスを呼び出します。サービスがプラグインに注入され、構築や依存関係の解決の負荷を IoC コンテナに移動します。2つ目は、プラグインがサービス出力境界インターフェイスを実装し、変更を永続化してデータベースに保存したり、応答を Web UI に送信したりなど、ユースケースの結果を処理します。

### 16.8.3. ドメインパッケージ

**domain** パッケージには、**business objects** が含まれており、**Location**、**Vehicle**、**Route** など、このプロジェクトのドメインをモデル化します。このようなオブジェクトは完全にビジネス指向で、オブジェクトリレーションマッピングツールや Web サービスフレームワークなど、ツールやフレームワークの影響を受けないようにする必要があります。

### 16.8.4. サービスパッケージ

**service** パッケージには、ユースケースを実装するクラスが含まれます。ユースケースには、新しい場所の追加、車両の積載量の変更、住所の座標検索など、実行することを記述します。ユースケースを統括するビジネスルールは、ドメインオブジェクトを使用して表現します。

サービスは、永続性、Web、最適化など、外層のプラグインを操作する必要があります。層と層の間の

依存関係ルールを満たすには、サービスの依存関係を定義するインターフェイスという観点で、サービスとプラグインの間のやり取りを表現します。プラグインは、サービスの境界インターフェイスを実装する Bean を指定して、サービスの依存関係を満たすことができます。CDI コンテナは、プラグイン Bean のインスタンスを作成し、ランタイム時にサービスに注入します。これは、制御原理の反転例です。

### 16.8.5. プラグインパッケージ

**plugin** パッケージには、最適化、永続性、経路、ネットワーク通信などのインフラストラクチャー機能が含まれます。

## 16.9. OPTAWEB 配送経路のバックエンド設定プロパティ

以下の表に記載されている OptaWeb 配送経路アプリケーションプロパティを設定できます。

プロパティ	タイプ	例	説明
<b>app.demo.data-set-dir</b>	相対パスまたは絶対パス	<b>/home/user/.optaweb-vehicle-routing/dataset</b>	カスタムデータセットは、このディレクトリーから読み込まれます。デフォルトは、 <b>local/dataset</b> です。
<b>app.persistence.h2-dir</b>	相対パスまたは絶対パス	<b>/home/user/.optaweb-vehicle-routing/db</b>	データベースファイルの保存に H2 が使用するディレクトリー。デフォルトは <b>local/db</b> です。
<b>app.region.country-codes</b>	<b>ISO 3166-1 alpha-2</b> 国コードの一覧	<b>US, GB,IE, DE,AT,CH</b> 。空白でも構いません。	地理検索結果を制限します。
<b>app.routing.engine</b>	列挙	<b>air, graphhopper</b>	経路エンジンの実装。デフォルトは <b>graphhopper</b> です。
<b>app.routing.gh-dir</b>	相対パスまたは絶対パス	<b>/home/user/.optaweb-vehicle-routing/graphhopper</b>	道路網グラフの保存に GraphHopper が使用するディレクトリー。デフォルトは <b>local/graphhopper</b> です。
<b>app.routing.osm-dir</b>	相対パスまたは絶対パス	<b>/home/user/.optaweb-vehicle-routing/openstreetmap</b>	OSM ファイルを含むディレクトリー。デフォルトは <b>local/openstreetmap</b> です。

プロパティ	タイプ	例	説明
<b>app.routing.osm-file</b>	ファイル名	<b>belgium-latest.osm.pbf</b>	GraphHopper が読み込む OSM ファイル名。 ファイルは <b>app.routing.osm-dir</b> に配置する必要があります。
<b>optaplanner.solver.termination.spent-limit</b>	<b>java.time.Duration</b>	<ul style="list-style-type: none"> <li>• 1m</li> <li>• 150s</li> <li>• P2dT21h (PnDTnHnMn.nS)</li> </ul>	場所の変更後にソルバーを実行する時間。
<b>server.address</b>	IP アドレスまたはホスト名	<b>10.0.0.123, my-vrp.geo-1.openshiftapps.com</b>	サーバーをバインドするネットワークアドレス。
<b>server.port</b>	ポート番号	<b>4000, 8081</b>	サーバーの HTTP ポート。

## 付録A バージョン情報

本ドキュメントの最終更新日: 2023 年 2 月 1 日

## 付録B お問い合わせ先

Red Hat Decision Manager ドキュメントチーム: [brms-docs@redhat.com](mailto:brms-docs@redhat.com)